Game Theoretic Verification of Timed Systems

First Annual Progress Seminar Report - Literature survey

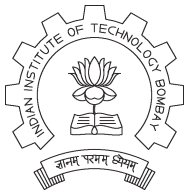
Submitted in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy
by

Lakshmi Manasa.G
Roll No: 08405002

under the guidance of

Prof. Krishna.S



Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai

Acknowledgments

I would sincerely like to thank **Prof. Krishna.S** for her consistent directions and their valuable suggestions.

Lakshmi Manasa.G

Abstract

Games are increasingly used to model open systems in both timed and untimed settings. The kind of game chosen depends directly to the system under consideration and the kind of question to be answered. The existence of determinacy, strategies for the players, the memory requirements of these strategies, the complexity of solving the game are some of the most sought after questions for untimed games. While for timed games there are additional concerns of time divergence, robustness, infinite memory requirements due to precise clocks and so on.

This report is to serve as a concise literature survey. We shall study basic notions of games, algorithms for untimed parity games and region based solutions of timed games. The focus shall be on the formulation of a game, the main result for that kind of game, interesting algorithms to solve the game and the complexity of solving it.

Contents

Chapter 1

Introduction

Verification has gathered a lot of importance in the past few decades owing to the need to be 100% correct in safety critical as well as mission critical systems. There are several techniques already in use and several more currently underway. Typically to verify a system, a formal model of the system and its specification are given and we are required to check whether the system meets its specification. Some of the most studied verification techniques include model checking wherein the system is given as a model while the specification is given as a logical formula, proof-based verification wherein both are described as formulae, asking simple analysis questions on the model of the system such as language emptiness, reachability, safety when a state-based model is used for the system. Several factors determine what kind of verification method we would choose for a problem at hand such as the kind of the system (open vs closed), kind of specification (full vs singular properties), complexity of the verification itself and so on.

Closed systems as the name suggests are independent entities not requiring regular interactions with their environments. On the other hand, open systems interact heavily with the environment. One would be quick to suggest considering the environment as a module of the system and consider it as a closed one but the spirit of the environment is very different from that of other modules. While the modules of the system co-operate to achieve a goal the same can not be extended to the loyalty of the environment towards the system. Hence, the kind of verification performed differs in spirit for closed and open systems. Model checking, simple language theoretic analysis questions might suffice for closed systems but for open systems we need to take into account the independence and the important role of environment. A very natural way to look at this setting would be as a game considering the system and the environment as two individual players in a game.

Games are becoming increasingly popular as a modeling paradigm for open systems in verification. Open system scenario arises very often in the synthesis of controllers due to the nature of the problem itself [44], [45]. Depending on the kind of verification to be performed, the game chosen could have finite or infinite plays, zero-sum or non-zero sum winning, strictly completive or coalitional interaction between the players, turn based or concurrently proposing their moves, perfection or imperfect information of the play available to the players and so on. The arena on which the game is played depends on the system under consideration such as timed or untimed, finite or infinite number of states

and so on.

Research in untimed games involves objectives such as reachability, safety, Büchi, Rabin, Streett and parity conditions over a graph arena. However, parity is the canonical form of expressing all the $\omega-$ regular acceptance conditions [2]. Also Rabin and Streett can be expressed as disjunctions and conjunctions of several parity conditions [20]. These results have led to parity games gaining a lot dedication from several researchers. The stage was set early on with [4] in '95 which gives a simple and elegant solution for parity games and proves positional determinacy. Another reason for the popularity of parity games is that they lie in the NP $\cap$ Co-NP bracket [23] with the existence of a polynomial algorithm for parity games being a long standing open problem. Attempts to improve the exponential algorithm given by [11] have resulted in numerous interesting ways of solving the problem. However, unfortunately the problem remains open with the best known algorithm having sub-exponential complexity [6].

Frequently asked questions in an untimed game setting include determinacy, improving the kind of strategies, improving the complexity of building a strategy with the latter gaining a lot of attention due to their coveted position in NP $\cap$ Co-NP bracket. The other dimension of growth here is exploring different kind of games like concurrent games with the same objectives, stochastic games, Borel games, priced games, mean-payoff games, mean-payoff parity games and so on. The missing concept of Nash-equilibrium in graph games is now gaining momentum with the notions of secure equilibrium [46], sub-game perfect equilibrium [47] being introduced.

The need for games in computer science research especially in the field of verification derives from the close relation between games and controller synthesis problems. With the introduction of timed automata [1] in '94 there has been an upsurge in the research of timed systems. Timed automata provide a very neat model which captures all the essential features at the same time maintaining the simplicity of semantics provide a finite syntax for the infinite systems. Closely following the introduction [35], [37] dealing with controller synthesis paved the for timed automata as arena for timed games. The recent researchers take a clearer approach to games on timed automata posing the well-known untimed $\omega-$regular conditions for timed automata as well. Apart from these there are average timed games played on timed automata which use time as a metric, average reward time games, priced games played on the weighted counterpart of timed automata [28].

The kind of answers sought in timed games differ little from those in untimed games. Due to the time dimension there are additional concerns of time divergence, robustness in strategies, dense time requirements in goals as opposed to the currently studied location-based $\omega-$regular goals, achieving true concurrency in a timed setting and so on. Though there has been some research to reduce the complexity of strategy building algorithms, it has not really gained much attention as most of the solutions rely on the untimed counterparts via classical region equivalence [1]. The complexity continues to be high due to the complexity of both region-based reduction to the untimed game and the complexity

of solving the untimed game itself. While this on one hand is a sad state of affairs, maintaining precise clocks to keep track of time by the strategies (typically implemented as controllers) results in infinite memory. Most objectives that enjoy positional memoryless determinacy in the untimed setting can get only as far as determinacy and at best finite memory determinacy if randomization is brought in to trade with infinite memory. There have been very few attempts, if any to solve a timed game at the timed level itself.

In this report, we shall study a handful of papers for both untimed and timed games to get ourselves familiar with the scenario in games. Chapter 2 introduces the basic concepts for games, some important results regarding two player zero-sum turn based games. For untimed games, we shall concentrate on parity games with 2 players, finite arena, infinite plays formed in a turn based fashion in the Chapter 3. As mentioned earlier, there have been several attempts at improving the complexity and we shall look at a few recent interesting ways of solving parity games. Timed games are explored in Chapter 4 where we look at timed parity games and a few alternate ways of solving them. We shall peek into both concurrent and turn based games to understand that concurrency can be removed in timed games [29]. In timed games, as most of them do not solve the timed game itself but aim for a result-preserving reduction, we will not have many strategy building algorithms to discuss. Hence, the focus is on the different kinds of games and their formulations with a brief idea about how they are solved. In both the chapter dealing with untimed games, we shall present the game, the algorithm to solve it and a brief calculation of complexity. In the timed games however, as we do not solve the timed game per-say, less emphasis is laid on the algorithm or the calculation of the complexity. We give a briefing of the algorithm and merely mention the complexity which is mainly dependent on the region automata size. We shall conclude in Chapter 5 with a few open problems derived from the games we have studied in the earlier chapters.

Chapter 2

Building blocks

Let S be a set. Let S^*, S^ω denote the set of finite and infinite words over S. If w is a word over S then $\mid w \mid$ denotes the length of the word. S^+ denotes all the finite words of non-zero length.

Graph

We adopt the usual notations for graphs. A *graph* G is denoted as $G = (V, E)$ where V is the set of vertices and $E \subseteq V \times V$ is the set of edges. These two sets V, E can be either finite or infinite. If both of them are finite then the graph is called a *finite graph*. A *path* π in a graph G is a sequence of vertices $\pi = v_0 v_1 v_2 \cdots v_n \cdots$ such that $\forall i,\ v_i \in V$ and $\forall i, (v_i, v_{i+1}) \in E$. A path is either finite $\pi \in V^*$ or infinite $\pi \in V^\omega$ depending on the length. G is referred to as a *directed graph* if the edge set E is considered as an ordered pairs of vertices otherwise it is called an *undirected graph*.

For a vertex $v \in V$, the set of its successors denoted as vE is $\{u \mid (v, u) \in E\}$. A vertex v with no successors $vE = \emptyset$ is called a *deadend*. Given a path $\pi = v_0 v_1 \cdots v_i \cdots$, the i^{th} vertex is denoted as π_i while the prefix π until π_i is denoted as $\pi[0-i]$. For a path $\pi = v_0 v_1 \cdots v_n \cdots$, $Occ(\pi) = \{v \mid \exists i, \pi_i = v\}$ is the set of vertices visited. For an infinite path $\pi = v_0 v_1 \cdots v_i \cdots$, $Inf(\pi) = \{v \mid \forall i, \exists j > i, \pi_j = v\}$ is the set of vertices visited infinitely often.

For the remainder of the report, we consider only finite graphs with infinite paths in them. However some of paths might be finite if they encounter a deadend. Vertices are alternately referred to as *states* and edges as *transitions*. Unless specified otherwise, this study pertains to undirected graphs.

2.1 Graph Games

A graph game $\mathcal{G}$ consist of a graph along with objectives for the players (> 1) of the game. The vertices are partitioned among the players who push a conceptual token when it is on their vertex along one of the outgoing edges to form an infinite path. Such an infinite path is called a *play* of the game. The formal definition is given below.

Definition 2.1. *A graph game is* $\mathcal{G} = (V, E, P, \{V_i\}_{i \in P}, \{O_i\}_{i \in P})$ *wherein*

1. *V is the finite set of vertices,*
2. *$E \subseteq V \times V$ is the finite set of edges,*
3. *P is the finite set of players,*
4. *$V_i \subseteq V$ is the set of vertices of player i. Note that the set V is partitioned among the players i.e; $\forall i, j,\ V_i \cap V_j = \emptyset$ while $\bigcup_i V_i = V$.*
5. *$O_i \subseteq V^\omega$ is the winning objective for the player i described as the set of plays which are favorable to her.*

Let Π denote the set of all plays in a game. In a play, *current state* is the vertex on which the token is at present. For simplicity of notation, we shall refer to the graph of a game as its *arena* denoted as $\mathcal{A}$. A vertex $v \in V_i$ is said to be a *deadend for a player i* if there is no outgoing edge i.e; $vE = \emptyset$.

Given a game $\mathcal{G}$, an *Initialized game* is a tuple $(\mathcal{G}, v)$ where $v \in V$. It takes into consideration all the plays which start from v.

Winning Objectives for players

A winning objective O_i for a player i is the set of plays that i considers favorable. Alternately they are also called as winning conditions. The different kinds of objectives studied in literature are given below.

We denote by $F_i \subseteq V$ ($\mathcal{F}_i \subseteq \mathcal{P}^V$) the set (family of sets) of states desirable to i. A play $\pi = v_0 v_1 v_2 \dots$ is said to be in O_i

- *Reachability* - iff $\exists i, v_i \in F_i$ i.e; atleast one of the desired states is visited during the play.
- *Safety* - iff $\forall i, v_i \in F_i$ i.e; all the states visited are desirable.
- *Büchi* - iff $Inf(\pi) \cap F_i \neq \phi$ i.e; atleast one of the desired states is visited infinitely often.
- Muller condition - iff $Inf(\pi) \in \mathcal{F}_i$ i.e; the set of states visited infinitely often is one of the desired sets.
- *Parity* - a coloring function is defined as $\chi : V \longrightarrow C$ where C is a set of priorities (which are usually integers) assigned to states of V. A play is in O_i iff the maximum (or minimum in some studies) priority in $Inf(\pi)$ is even.

A player loses a play $\pi \notin O_i$ if the play ends in a deadend $v \in V_i$.

Strategy for a player

There are several techniques for a player to ensure that the play is favorable to her. A strategy for a player is a guiding light for her which tells at each step which vertex to choose next so as to turn the play favorable. A strategy might require a lot memory to remember every move since the start or might be able to pick the next vertex considering just the current state.

Definition 2.2.
- *A strategy for player i is $\sigma_i : V^*V_i \to V$. Given a finite prefix of a play ending in a vertex belonging to player i, σ_i suggests the next vertex to be chosen so that the play turns out to be favorable to her.*
- *A strategy for player i is said to be winning in a vertex $v \in V$ if any play starting at v agreeing with σ_i is in O_i irrespective of the actions of the other players.*
- *A set $W_i \subseteq V$ is called the Winning set (or winning region) for player i if she has a winning strategy in all the vertices of W_i.*

We denote by Σ_i the set of all strategies for player i. If player i decided to use strategy $\sigma_i \in \Sigma_i$, the resulting play starting at state sv is represented as $outcome(v, \sigma_0, \sigma_1, \cdots)$. Hence σ_i is winning for i from state v iff $outcome(v, \sigma_0, \sigma_1, \cdots) \subseteq O_i$ for all $\sigma_j,\ j \not= i$.

There are different kinds of strategies. A strategy σ_i is called

1. memoryless strategy iff $\sigma_i(wv) = \sigma_i(w'v)$ where $w, w' \in V^*$;
2. counting strategy iff $\sigma_i(w) = \sigma_i(w')$ wherein $w, w' \in V^*$ and $\mid w \mid = \mid w' \mid$;
3. finite-memory strategy iff there is another function which modifies the finite memory as the play progresses and aids σ_i in deciding the next state.
4. infinite memory strategy iff the memory involved is infinite.

A game is called *zero-sum* if $\forall i, j,\ O_i \cap O_j = \emptyset$. These games are competitive where each player i tries to ensure that the resulting play belongs to O_i. It is easy to see that for a zero-sum game, there is atmost one winner for a given play i.e; $\forall i, j,\ W_i \cap W_j = \emptyset$. A zero-sum game is said to be *determined* if for a given vertex $v \in V$, one of the players wins the play starting at v i.e; $\bigcup_i W_i = V$. If we consider a two player game, it is clear that the winning objective of a player is a complement of the other player. Due to this, they are often referred to as opponents of each other.

If all the players in a determined game have memoryless strategies then the game is said to enjoy *memoryless determinacy*. Similar notion is extended to all the other kinds of strategies studied above.

Consider a 2-player zero-sum game $\mathcal{G} = (V_0, V_1, E, O_0, O_1)$. If a winning condition is specified for player 0 and the game is zero-sum then it is clear that the winning condition of 1 is the complement of the condition for 0. We refer to a 2-player zero-sum game as a reachability game if the winning condition for 0 is a reachability condition and similarly for other conditions too.

2.2 2-player zero-sum graph games

In this subsection, we restrict ourselves to some basic concepts pertaining to 2-player zero-sum graph games. However, it is to be noted that most of these concepts can be easily extended to multi-player zero-sum games. These concepts are central to the nature of graph games and hence independent of the winning conditions. Let the two players be 0 and 1.

2.2.1 Attractor and trap sets

Given a game $\mathcal{G}$ with arena $\mathcal{A} = (V, E)$ and a set $X \subseteq V$, player i can force a visit to the set X in zero or more moves from the current state of the play. The set $Attr_i(\mathcal{G}, X)$ called the *Attractor set of player i for set X* is the set of vertices from which i can force a visit to some vertex in X irrespective of the wishes of the opponent. Before the construction of the attractor set let us look into some simple yet important operators.

For some $Y \subseteq V$, $pre_i(Y) = \{v \in V_i | vE \cap Y \neq \phi\} \cup \{v \in V_{1-i} | vE \subseteq Y\}$.

The following is the construction of $Attr_0(\mathcal{G}, X)$ for some $X \subseteq V$.
$Z_0 = X$
$Z_i = Z_{i-1} \cup pre_0(Z_{i-1})$
$Attr_0(\mathcal{G}, X) = Z_i$ where i is the smallest value such that $Z_i = Z_{i+1}$

Note that the construction of $Attr_1$ is similar. The example 2.2.3 gives the construction of an attractor set.

Rank r assigns to each vertex s in an attractor set $Attr_i(\mathcal{G}, X)$ a positive integer indicating the length of the shortest path from s to a vertex in X. Formally, $r(s) = 0$ if $s \in X$, else $r(s) = j$ such that $s \in Z_j \setminus Z_{j-1}$. A memoryless strategy σ_i to reach X from any vertex in $Attr_i(\mathcal{G}, X)$ is given as $\sigma_i(s) = s'$ such that rank of s' is lower than that of s.

A *Trap for player i* is a set T such that she cannot force a visit to $V \setminus T$. Attractor and trap concepts are duals of each other as we shall prove below. A set $X \subseteq V$ is called as a *0-paradise* iff it is a trap for 1 and 0 has a memoryless winning strategy in X. Following are a few easy to prove lemmas about attractors.

Lemma 2.1. *Let $\mathcal{G}$ be a game with arena $\mathcal{A}$ =(V, E). Then*

1. *The set $V \setminus Attr_i(\mathcal{G}, X)$ is a trap for player i.*
2. *The set $Attr_i(\mathcal{G}, X)$ is a trap for player $1 - i$.*
3. *If U is a 0-paradise then so is $Attr_0(\mathcal{G}, U)$*
4. *A union of 0-paradises is also a 0-paradise*

Note that the same is true for 1-paradises. [[2] lemma 6.5]

Proof. 1. Assume that $V \setminus Attr_i(\mathcal{G}, X)$ is not a trap for player i. Then $\exists v \in V \setminus Attr_i(\mathcal{G}, X)$ such that i can force a visit to $V \setminus \{V \setminus Attr_i(\mathcal{G}, X)\} = Attr_i(\mathcal{G}, X)$. But if such a v did exist then by definition of $Attr_i(\mathcal{G}, X)$, $v \in Attr_i(\mathcal{G}, X)$, a contradiction. Hence $V \setminus Attr_i(\mathcal{G}, X)$ is a trap for i.

2. Assume that $Attr_i(\mathcal{G}, X)$ is not a trap for $1 - i$. Then $\exists v \in Attr_i(\mathcal{G}, X)$ such that $1 - i$ can force a visit out of $Attr_i(\mathcal{G}, X)$. But by construction of $Attr_i(\mathcal{G}, X)$ (recall the definition of operator pre_i), $v \notin Attr_i(\mathcal{G}, X)$, a contradiction. Hence $Attr_i(\mathcal{G}, X)$ is a trap for $1 - i$.

3. As U is a 0-paradise 0 has a memoryless winning strategy from U. Now $Attr_0(\mathcal{G}, U)$ is the set of states such that 0 can force a visit to U by using a memoryless rank-based strategy. Once in U, 0 plays the memoryless strategy that she has in U. Additionally from the above argument it is clear that $Attr_0(\mathcal{G}, U)$ is a trap for 1 and by definition of 0-paradise U is also a trap for 1. Hence $Attr_0(\mathcal{G}, U)$ is a 0-paradise.

4. Consider a union U of 0-paradises. It is clear that 1 does not have path from this set to $V \setminus U$ [refer lemma 2.3] and thus U is a 1-trap.
Now the memoryless strategy can be decided as follows, an arbitrary ordering can be given to all the 0-paradises such that they are X_i where $i \in \{1, 2, 3..\}$ and then for a vertex $v \in X_i$ where i is the smallest such value possible, 0 chooses the strategy meant for X_i as long as it is in X_i and wins. But if the play ventures out of X_i then it has to be to another 0-paradise only [otherwise it would not be a 1-trap] and in this new paradise 0 follows the strategy of this paradise and so on. This would be the memoryless winning strategy for 0 in U. Hence U is also a 0-paradise.

□

2.2.2 Subgame

Given a set $X \subseteq V$, the graph induced by X is called a subgame denoted by $\mathcal{G}[X]$ such that

1. the arena $\mathcal{A}_X = (V \cap X, E \cap (X \times X))$
2. every deadend in $\mathcal{G}$ is also a deadend in $\mathcal{G}[X]$ which means that the subgame cannot introduce any new deadends.

Lemma 2.2. *Let U and U' be subsets of V such that $\mathcal{G}[U]$ is a subgame of $\mathcal{G}$ and $\mathcal{G}[U][U']$ is a subgame of $\mathcal{G}[U]$ then $\mathcal{G}[U']$ is a subgame of $\mathcal{G}$. [book [2] lemma 6.2]*

Proof. The subgame $\mathcal{G}[U']$ satisfies the three conditions required for it to be a subgame of $\mathcal{G}$. ie; $U' \subseteq V$ and the arena of $\mathcal{G}[U']$ would be as required [because of the transitivity of set inclusion] and also no new deadends would be introduced as $\mathcal{G}[U']$ is a subgame of $\mathcal{G}[U]$. □

Lemma 2.3. *1. For every 0-trap U in $\mathcal{G}$, $\mathcal{G}[U]$ is a subgame.*

2. For every family of 0-traps, their union is also a 0-trap.

3. *If X is a 0-trap in $\mathcal{G}$ and Y is a subset of X, then Y is a 0-trap of $\mathcal{G}$ iff Y is a 0-trap in $\mathcal{G}[X]$.*

Note that same results hold for 1-traps. [book [2] lemma 6.3]

Proof. 1. Given a 0-trap U, to prove that $\mathcal{G}[U]$ is a subgame all we need to show is that a deadend in $\mathcal{G}[U]$ would also be a deadend in $\mathcal{G}$. The proof of this claim follows. Consider a deadend v in $\mathcal{G}[U]$ which is not a deadend in $\mathcal{G}$.
Now if $v \in V_0$ then there would be an edge going out of v to $V \backslash U$ in $\mathcal{G}$ which means that in $\mathcal{G}$ 0 has a chance to escape out of U ensuring that U is not a 0-trap, a contradiction.
Now if $v \in V_1$ and $vE \neq \phi$ in $\mathcal{G}$ then it would become a deadend in $\mathcal{G}[U]$ iff $vE \subseteq V \backslash U$. But this means that all the outgoing edges from v lead out of the trap, enabling 0 to ensure that the play eventually moves out of U. Then U is not a 0-trap at all, which is a contradiction.

2. A 0-trap U is a set of states from which 0 does not have a path to $V \backslash U$ and thus cannot escape U. A union of family of 0-traps will also be a trap for 0 because 0 cannot escape from any of these individually and 0 does not have a path from any of them to the states outside them. Now, consider a family of 0-traps $U_0, U_1, U_2 \cdots$ and let the union of all the 0-traps U_i be X. Assume that X is not a 0-trap allowing 0 to have a path from X to $V \backslash X$. Then there exists a vertex v in some 0-trap U_i such that $(v, v') \in E$ and $v' \notin X$ i,e; $v' \in V \setminus X$. But as $U_i \subseteq X$, $v' \in V \setminus U_i$. But this would mean that U_i is not a 0-trap, a contradiction. Thus X is a 0-trap.

3. Previously it has been proved that a 0-trap X is a subgame. $Y \subseteq X$ is a 0-trap in $\mathcal{G}$ means that 0 doesn't have a path from Y to $V \backslash Y$. Now this also means that 0 doesn't have a path from Y to $X \backslash Y$ as $Y \subseteq X \subseteq V$, which implies that Y is a 0-trap in $\mathcal{G}[X]$. Similar argument holds other way round too.

□

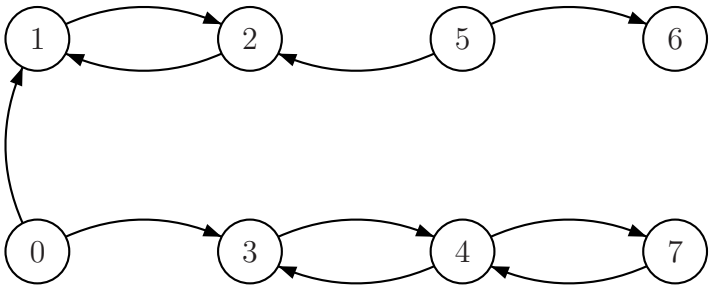
2.2.3 Reachability Games

As has been defined earlier, a reachability game is the one in which the winning condition specifies that a play is winning for player 0 iff atleast one of the desired states in visited during the play. We denote reachability games as $\mathcal{G} = (V_0, V_1, E, F)$ where F is the desired set for the reachability condition of 0.

Theorem 2.1. *Reachability games have memoryless determinacy (ie both the players have memoryless strategies) [book [2] proposition 2.18]*

Proof. Given a game $\mathcal{G} = (V_0, V_1, E, F)$, the set $Attr_0(\mathcal{G}, F)$ is constructed for player 0. By definition of $Attr_0(\mathcal{G}, F$, player 0 can force the play into a state in F in zero or more steps from any state $v \in Attr_0(\mathcal{G}, F)$. Thus 0 can win the infinite plays from $Attr_0(\mathcal{G}, F)$.

However, to win finite plays 0 has to ensure that the play terminates in a deadend in V_1. This can be done by constructing another attractor set such that in zero or more steps



Example 2.1.

Figure 2.1: Game arena for reachability game

0 can force the play to terminate in a deadend of player 1. But the interesting point to be noted is that for a deadend $v \in V_1$, $vE = \phi \subseteq Z_0$ which means that by definition of pre_0, all the deadends belonging V_1 must have been included in Z_1 during the construction of $Attr_0(\mathcal{G}, F)$. [refer to section 2.2.1]. The winning region for player 0 is $W_0 = Attr_0(\mathcal{G}, F)$ and the winning strategy σ_0 is defined as the memoryless rank-based strategy described in section 2.2.1.

The set $W_1 = V \setminus W_0$ is a trap for 0 as 0 cannot force any play originating in W_1 to visit F. Hence W_1 is winning for player 1. The memoryless winning strategy σ_1 arbitrarily chooses one of the successors for a vertex i.e; $\forall v \in W_1$, $\sigma_1(v) = u \in vE$.

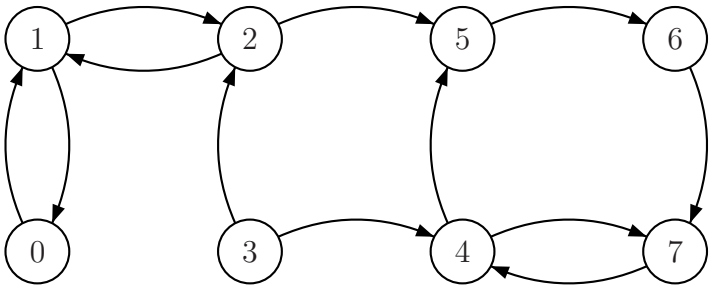
As both the players have memoryless winning strategies and as the sets $W_0 \cup W_1 = V$, reachability games enjoy memoryless determinacy. □

Consider the arena given in the figure 2.1. V_1 is all the odd numbered states while the rest form V_0 and the desired set is $F = \{6, 7\}$. The construction of the attractor set $Attr_0(\mathcal{G}, F)$ (as has been explained the previous chapter using same symbols as in the theorem for this construction) would be as follows.
$Z_0 = F = \{6, 7\}$
$Z_1 = \{6, 7, 4\}$
$Z_2 = \{6, 7, 4, 3\}$
$Z_3 = \{6, 7, 4, 3, 0\}$
$Z_4 = \{6, 7, 4, 3, 0\}$
$W_0 = Attr_0(\mathcal{G}, F) = Z_4$.
It is easy to note that 0 can not force a visit to the desired set from any other state.

2.2.4 Büchi Games

Similar to reachability games, Büchi games have a desired set F such that π is winning for 0 iff $Inf(\pi) \cap F \neq \emptyset$ or ends in a deadend for 1. A Büchi game is represented as $\mathcal{G} = (V_0, V_1, E, F)$.

Theorem 2.2. *Buchi games enjoy memoryless determinacy. [Theorem 2.22 [2]]*



Example 2.2.

Figure 2.2: Game arena for Buchi wining condition

Proof. In a Buchi game, the winning condition for player 0 is $Inf(\pi) \cap F \neq \phi$ or the last state v of π is such that $v \in V_1$ and $vE = \phi$.
Given this information, let us first construct $Attr_0(\mathcal{G}, F)$, the attractor set of 0 for the set F such that 0 can force the play to visit F in 0 or more moves. Recall that this set would include the deadends of 1. This accomplishes a part of W_0 construction.

Now we have to find a subset $Recur(F)$ of $Attr_0(\mathcal{G}, F)$ from which 0 can force a visit to atleast one state of F atleast one more time. This set would be the winning region for 0 because from this set she can force a visit to F infinitely often. So, $W_0 = Recur(F)$.

The construction of $Recur(F)$ is as follows
$Z_0 = F$ - desired states
$X_i = Attr_0(\mathcal{G}, Z_i)$ - states from which 0 can force a visit to Z_i in 0 or more moves
$Y_i = pre_0(X_i)$ - states from which X_i can be visited in exactly 1 step.
$Z_{i+1} = Y_i \cap F$ - states in F from which 0 can force a visit to F exactly one more time.
Let k be the least such value such that $Z_k = Z_{k+1}$, then let $U = \cup_{i \leq k} Z_i$
$Recur(F) = Attr_0(\mathcal{G}, U)$.
Note that the deadends of player 1 are included in $Recur(F)$ as it is an attractor set for player 0 and by definition of pre_0 all deadends of 1 will be included.
$W_0 = Recur(F)$ is the winning region of 0. The memoryless strategy σ_0 for player 0 is defined as follows. For a play $\pi = v_0 v_1 v_2 .. v_i$ then the next state is $\sigma_0(v_i) = v_{i+1}$ chosen as explained below.

- If $v_i \in V_1$ and is a deadend then the play is finite and it is a win for player 0.
- If $v_i \in V_0 \cap W_0$ then $\exists v' \in vE | v' \in W_0$ [by definition of $Recur(F)$]. $v_{i+1} = v'$ as it ensures that atleast some state of F is visited infinitely often.
- If $v \in V_1 \cap W_0$ then any state $v' \in vE$ can be chosen because $vE \subseteq W_0$.

Using an argument similar to the proof of Theorem 2.1 we know that player 0 cannot win from all states in $V \backslash W_0$. Thus $W_1 = V \backslash W_0$ is the winning region for player 1. The winning strategy for player 1 is symmetrical to σ_0. □

Consider the Arena in figure.2.2. V_1 is all the odd numbered states while the rest form V_0 and the desired set $F = \{5, 6, 7\}$ for the Buchi winning condition. Here the set W

would be $\{4,5,6,7\}$ *while the set* $W_0 = Recur(F) = \{2,3,4,5,6,7\}$. *It is to be noted that from any other state 0 cannot win the play.*

2.2.5 Parity Games

In parity games, there is a coloring function $\chi : V \longrightarrow C$ where $C \subset \mathbf{Z}^+$ is a set of priorities. For a play $\pi = v_0v_1v_2 \ldots$. $\chi(\pi) = \chi(v_0)\chi(v_1)\chi(v_2)$. The winning condition for player 0 requires $Max(\chi(Inf(\pi)))$ to be even [or as with every other game, the finite play terminates in a deadend for player 1].

The following theorem proves that parity games also enjoy memoryless determinacy.

Theorem 2.3 (Memoryless determinacy of parity games)**.** *The set of vertices of a parity game is partitioned into a 0-paradise and a 1-paradise. [Theorem 6.6 [2]]*

Proof. This proof is built on the induction of the maximum priority n.
Base case : Consider an arena with $n = 0$. Here player 1 can win only on the deadends of 0. Thus the winning region $W_1 = Attr_1(\mathcal{G}, \phi)$ and the winning region of player 0 is $V \backslash W_1$. The memoryless strategies are pretty straight forward.

Now assume that $n \geq 1$. By induction assume that there are memoryless strategies for both players for games with maximum priority less than n.
Let X_1 be a 1-paradise in $\mathcal{G}$. It is called as a trap for 0 because from any of these states 0 cannot force the play such that the maximum priority repeated is even [otherwise there would not be a winning strategy for 1 from this set]. Now it is clear that $X_0 = V \backslash X_1$ is a 1-trap.
Also define the sets $N = \{v \in X_0 \mid \chi(v) = n\}$ ie; all those states in X_0 whose priority is n. Let $Z = X_0 \backslash Attr_0(\mathcal{G}[X_0], N)$ all those states in X_0 from which 0 cannot force a visit to N.
As X_0 is a 1-trap, $\mathcal{G}[X_0]$ is a subgame of $\mathcal{G}$ [by lemma 2.3] and as Z is a complement of an attractor set in $\mathcal{G}[X_0]$ it is a 0-trap and hence $\mathcal{G}[X_0][Z]$ is also a subgame of $\mathcal{G}[X_0]$ and is a subgame of $\mathcal{G}$ [by lemma 2.2]. Now it is to be noted that Z does not contain any state with priority n and will have priorities $\{0, 1, ...n-1\}$ and by induction hypothesis, this game is partitioned into 0 and 1 paradises denoted as Z_0 and Z_1 respectively. Now Z_1 is a 0-trap in $\mathcal{G}[Z]$ and Z is a 0-trap in $\mathcal{G}[X_0]$ and thus by lemma 2.3 Z_1 is a 0-trap in $\mathcal{G}[X_0]$ Thus X_1 and Z_1 are 0-traps in $\mathcal{G}$ [be lemma 2.3]. The reason for this is basically, within Z_1 1 has a strategy because of which, 0 cannot escape from the set and the play continues to be within Z_1 and similarly within X_1. This would basically mean that 1 has a memoryless winning strategy within $X_1 \cup Z_1$ which also happens to be a 0-trap [by lemma 2.3] and thus this union is a 1-paradise.
However if Z_1 is empty which basically means that $X_0 = Z_0$ then X_0 is a 0-paradise because 1 has no escape out of Z_0 as it is the winning region of 0 in the subgame $\mathcal{G}[Z]$ and also 0 has a memoryless winning strategy in Z_0 and thus it would be a 0-paradise.
Now all we have to prove is that this X_0 is indeed a 0-paradise. To do that let us first take a family of all the 1-paradises there are in $\mathcal{G}$ and denote it as $\mathcal{W}_1$ and let the biggest such paradise be W_1 and this would be an union of all the paradises that belong to $\mathcal{W}_1$.

Now $Attr_1(\mathcal{G}, W_1)$ is the attractor set for 1 and this would again be a 1-paradise [refer lemma 2.1]. But as W_1 is the largest such 1-paradise possible $Attr_1(\mathcal{G}, W_1) = W_1$. Now considering $W_0 = V \backslash W_1$ this is a 1-trap as it is a complement of an attractor set and applying the same logic as we did with $X_1 = W_1$ and $X_0 = W_0$ previously, we obtain that $W_1 \cup Z_1$ is a 1- paradise. But as W_1 is a union of all the 1-paradises there are and $Z_1 \not\subseteq W_1$, it follows that $Z_1 = \phi$. Now this leaves $X_0 = W_0$ as the 0-paradise and thus we have shown that the parity game $\mathcal{G}$ is partitioned into 0 and 1 paradises. □

2.3 Other infinite games - Generalised Muller Games

In this section, we consider infinite games played on finite graphs with the modifications that

1. there are no deadends meaning that $\forall v \in V, vE \neq \phi$ and hence there are no finite plays

2. the winning condition is defined as follows, there is a set $U \subseteq V$ and a family of sets $\Omega \subseteq \mathcal{P}^U$ and for a play π $Inf(\pi)$ is the subset of U repeated infinitely often. Player 0 is said to win a play π iff $Inf(\pi) \in \Omega$. The important point to note here is that if $U = \phi$ then $\Omega = \{\phi\}$ and 0 always wins and if $\Omega = \phi$ then 1 always wins. For that matter 0 wins whenever we can decide that $Inf(\pi) = \phi$.

Before describing the results for these games, we introduce a finite memory strategy that depends on which states of U have been visited and this would help us decide which state to choose next so that 0 wins the play. This finite memory called *Last visitation record (LVR)* is defined as follows. If the current state of the play is $v \in U$ and the LVR before the current move is $LVR = w_1 w_2 ... w_h$, then one of the two possible cases are that

1. v has never been visited before in which case it is appended to LVR which now becomes $LVR = w_1 w_2 .. w_h v$

2. or v has been visited before and $v = w_j$ in which case the new LVR would be $LVR = w_1 w_2 .. w_{j-1} w_{j+1} .. w_h v$

It is very easy to observe that these games are a more general version of *Muller* games. The results for these games can be extended to Muller games by taking $U = V$ and then Ω will be the family of sets one of which must be repeated infinitely often for 0 to win the play. Also note that there is no chance that $Inf(\pi)$ becomes a null set in such a case because U is now V.

All the proofs that are to be discussed with regard to these games, use only the information in LVR and thus are finite memory games. The strategies used in these games are called LVR strategies which in simple words mean that the strategy σ_i is a function which takes into account the current state and the current value of LVR and decides the next state so that i wins the play. It is very easy to note that memoryless strategies are a special case of LVR strategies where in the strategy picks the same state as the successor irrespective of the LVR.

Simple results

We shall now discuss some simple results that hold for LVR strategies.

Theorem 2.4. *If a player of a game $\mathcal{G}$ has LVR strategies f and g such that, for sets $A, C \subseteq V$, f is a winning LVR strategy from A and similarly g from C, then there is a winning LVR strategy h for that player from $A \cup C$. Furthermore, if f and g are no-memory strategies then so is h.[paper [4] theorem 3.1]*

Proof. The proof to this is very similar to the proof of lemma 2.1 and as the proof does not use the fact that paradises use no-memory strategies, the same proof holds good here too. Infact, this theorem can be extended to include union of several sets from which a player has winning LVR strategies. □

Theorem 2.5. *If a player of a game has a winning LVR strategy f playing from $p \in V$ and $q \in V$ is visited in a play from p in which the player always follows f, then he has a winning LVR strategy f' playing from q. Furthermore, if f is a no-memory strategy then so is f'. [paper [4] theorem 3.2]*

Proof. Let A be a finite portion of the play π till q which started from p and played according to f. Now f is a LVR strategy and would look into the LVR at the state before q before deciding that q should have been taken. Now we can device another LVR strategy f' such that it takes into consideration the new LVR' obtained from the previous LVR but with all the states belonging to both U and A deleted. And this strategy could be designed such that it mimics f ie; it choses the same state after q as f would have chosen given the previous LVR. Then, Inf sets of both the plays would be the same and thus player would win even if he started from q. □

A subgraph induced by $X \subseteq V$ which is $(X, V_0 \cap X, V_1 \cap X, E' = E \cap (X \times X))$ and $U' = U \cap X$ and $\Omega' = \Omega \cap \mathcal{P}^{U'}$ is called a subgame of $\mathcal{G}$ iff $\forall p \in X \exists q \in X$ such that $(p, q) \in E'$. And any winning play for a player in this subgame is also a winning play in $\mathcal{G}$. It is straight forward that the subgraph is a subgame by the definition of the subgraph and the winning play has a $Inf(\pi)$ belonging to Ω' which also belongs to Ω.

Theorem 2.6. *For any $X \subseteq V, \phi \neq X \neq V$, if one of the players has a strategy from X to keep the play within X forever then the game graph induced by X is a subgame.*

Proof. Let 0 be the player who has a strategy to keep the play within X. This means that atleast one of the 0-states has a successor within X and all of 1-states have their successors within X. And this means that $\forall p \in X \exists q \in X$ such that $(p, q) \in E'$ and the other conditions needed are already fulfilled and hence this is a subgame. □

2.3.1 Forgetful determinacy

Now we show that the games under consideration have forgetful determinacy i.e; both the players have winning LVR strategies.

Theorem 2.7. *In a game $\mathcal{G}$, $W_0 \cap W_1 = \phi$ and $W_0 \cup W_1 = V$ and player i's winning LVR strategy from W_i can be determined from $\mathcal{G}$. [Theorem 4.1 [4]]*

Proof. Let $|V| = m$ and $U = \{w_1, w_2...w_n\}$. Now if $U = \phi$ then as was already noted 0 always wins as the $Inf(\pi) = \phi \in \Omega$. So, the following proof is for $U \neq \phi$. It is based on the induction of m. If $m = 2$ and note that it cannot be less than 2 by the definition of a game given earlier then there are only two things to consider irrespective of what U is 0 wins if $U \in \Omega$ else 1 wins. Now we consider games with $m \geq 3$ and assume that the theorem holds for games with $m < 3$.
For each $i \leq n$, a set $N_{i,0}$ and $N_{i,1}$ are constructed such that 0, 1 can keep out of w_i forever playing from $N_{i,0}$ and $N_{i,1}$ respectively. Now, $N_0 = Attr_0(\mathcal{G}, \cup_{i \leq n} N_{i,0})$ is the set from which 0 can play such that he keeps out of U forever and wins as $Inf = \phi \in \Omega$. Similarly $N_1 = Attr_1(\mathcal{G}, \cup_{i \leq n} N_{i,1})$ is the set from which 1 can win.
At this stage we have to consider a smaller set of V which is $V - N_0 - N_1$. there are three cases that are possible. Each of these have been discussed below.

1. $V - N_0 - N_1 \neq \phi$ and $N_0 \cup N_1 \neq \phi$
   0 cannot move out of $V - N_0$ (otherwise that state would be in N_0) which means 1 can keep the play within $V - N_0$ forever and by theorem 2.3 the graph $\mathcal{G}_1 = V - N_0 = V_1$ is a subgame of $\mathcal{G}$. Also, from a subset of V_1 which is $V' = V_1 - N_1$ 1 cannot escape which means that this set induces a subgame $\mathcal{G}'$ within $\mathcal{G}_1$. This subgame $\mathcal{G}'$ is also as subgame of $\mathcal{G}$ [by transitivity]. Thus $\mathcal{G}'$ is a subgame with number of states $< m$ and thus by induction has $W_1^{'}$ and $W_0^{'}$ along with required strategies. These strategies can be extended to strategies in $\mathcal{G}$ without any modification because moving out of $\mathcal{G}'$ would mean a loss to the players because to move out of this 0 has to move to N_1 [$V' = V - N_0 - N_1$] as it cant go to N_0 and thus loses the play. Given these sets and strategies we can say that $N_0 \cup W_0^{'} = W_0$ and the strategy built according to the theorem 2.3.

2. $V - N_0 - N_1 = \phi$. In which case, the process terminates and $W_0 = N_0$ and $W_1 = N_1$ for which there are memoryless strategies as these are attractor sets.

3. $N_0 \cup N_1 = \phi$. In this case, we prove that 0 always wins if $U \in \Omega$. else 1 wins. If $U \in \Omega$ then 0 wins if all the states of U are repeated infinitely often. For this we build the following sets.

   - $A_i = Attr_0(\mathcal{G}, w_i)$ and from this 0 has a no memory strategy as has been seen in the previous chapters.
   - $V_i = V \backslash A_i$. This is a subgame $\mathcal{G}_i$ of $\mathcal{G}$ because 0 cannot escape from this and all the states which have edges from V_i to A_i are 1 states and for all states within V_i there is atleast one other state $\in V_i$ such that there is an edge between them. In this subgame 1 has no winning strategy because if he had any such strategy then let $W_{i,1}$ be a set such that he has a winning strategy which basically means that he can play such that he can keep the play within $\mathcal{G}_i$ forever and win and thus keep it out of w_i and this would mean that $W_{i,1} \subseteq N_1$ but $N_1 = \phi$ and thus 1 has no strategy to win in this subgame.
   - Now, this subgame has lesser than m states and by induction hypothesis there should be a winning strategy f for 0. This strategy is keep the play within V_i as long as 1 allows it (because only 1-states have a transition from V_i to A_i)

else if 1 moves out of V_i necessarily to A_i then force a visit to w_i using the memoryless strategy. [It is to be noted that the existence of this strategy does not in any way contradict $N_0 = \phi$ because this strategy cannot help 0 keep the play out of w_i forever, it can only help 0 choose such that if 1 decides to be in V_i then stay within V_i else visit w_i]

- To win the play when 1 moves out of V_i, 0 has to ensure that all the states of U are repeated infinitely often. For this purpose, he maintains a number $\theta(LVR)$ which indicates which state of U to be visited next to win the play. This number would indicate an unvisited state of U (we can have an ordering within U such that without confusion this number points to the smallest such state possible) else if all have been visited then point to the first state of LVR such that it is revisited. The strategy for 0 is, if the current state is $v \in A_i$ and $\theta(LVR) = i$ then use the memoryless strategy to visit w_i and update $\theta(LVR)$ and if $v \in V_i$ then choose the next state according to $f(L', v)$ where $L' = LVR$ but with all the states in $U \backslash V_i$ deleted. Now proceeding like this, the $Inf(\pi)$ is either ϕ if 1 allows the play to be within V_i forever else U if 1 decides to leave V_i.

We have yet to prove the other part of the theorem that the sets W_0 and W_1 and thus N_0 and N_1 are constructible. This will be proved now. Given $N_{i,0}$ building N_0 is easy. To build $N_{i,0}$ for w_i we do the following

- $X_i = Attr_1(\mathcal{G}, w_i)$. And for all $v \in V - X_i$ there atleast one other state $q \in V - X_i$ such that there is an edge from v to q (else $p \in X_i$). This means to say that 0 can keep the play within $V - X_i$ forever and thus it is a subgame by theorem 2.3.

- Now $Q = V - X_i$ has lesser than m states $w_i \notin Q$ then by induction, there should be Z_0 and Z_1 such that i has a winning strategy as claimed by the theorem. Also as 1 cannot go out of this Q (if it could then it would belong to X_i), 0 can keep the play within Z_0 forever in $\mathcal{G}$ also.

This Z_0 itself is $N_{i,0}$ because this gives 0 a strategy to keep out of w_i forever and win. Now similarly $N_{i,1}$ can also be constructed. □

2.3.2 Memoryless pair

A pair (U, Ω) is called a *no-memory pair* if U is finite and all games (V_0, V_1, E) with these as their wining conditions are no-memory games (which means that one of the players has a no-memory strategy).
A pair (U, Ω) is said to have a *split* if there exist $\alpha, \beta \subseteq U$ such that either $\alpha \cup \beta \in \Omega$, $\alpha \in \mathcal{P}^U$ and $\beta \in \mathcal{P}^U$ or else $\alpha cup \beta \in \mathcal{P}^U$,$\alpha \in \Omega$ and $\beta \in \Omega$.
A pair (U, Ω) is a no-memory pair iff it has no splits [paper [4] theorem 6.2]

We are now familiar with some of the basic concepts in graph games. The focus for the rest of the report will remain concentrated on different kinds of two player games and unless mentioned otherwise we would be dealing with zero-sum winning conditions. Henceforth, we shall dedicate one chapter each for a different kind of game like parity games, concurrent games where both players propose their moves simultaneously, graph

games enriched with timing information and weighted games where different vertices and edges are perceived differently by the players. Also for the sake of simplicity of notation, we will drop some of the components of a tuple whenever it is clear from the context.

Chapter 3

Parity Games (Untimed)

We shall now focus on two player zero-sum turn based games with a finite arena and infinite plays. Each vertex in the arena has a priority assigned to it. The winning condition for player 0 is to ensure that the maximum priority repeated infinitely often is even. As the game is zero-sum, player 1's winning objective is to make the maximum priority odd. Some of the algorithms are designed for the minimum priority. However, it is easy to see that these can be adopted to maximum priority with minimal changes.

Parity condition is the objective which has attracted a lot of attention amongst all the ω-regular objectives seen in the Chapter 2. This is so because of the polynomial reduction of any given ω-regular objective on a finite arena to a parity objective on a finite arena [2]. Due to this reduction, it is sufficient to study only parity objectives.

From the Theorem 2.3, it follows that all games with parity objectives are determined and that both players have memoryless strategies which help them achieve their goal. The complexity of algorithms building these strategies has been a topic of intense research because of their coveted position in the $NP \cap Co - NP$ bracket. Existence of a polynomial time algorithm for parity games has been the holy grail of this research but the best known algorithm has sub-exponential complexity. Several failed attempts at reaching the polynomial target have resulted in a series of interesting construction techniques for strategy algorithms. We shall familiarize ourselves with these techniques in the coming sections of this chapter. The proofs of termination, correctness and time complexity are beyond the scope of this report.

The table 3.1 summarizes some of the interesting results for parity games. Unless specified otherwise the arena is finite with undirected edges and vertices partitioned between the two players. An infinite play is the result of player pushing the token along one of the outgoing edges when the token is on their vertex. Also the arena is assumed to have no deadends. The winning objective for player 0 is that the maximum priority repeated infinitely often is even. The case with deadends is easy to deal with. We shall mention any deviations in description of the game under consideration.

We shall study each of these results in detail in the coming subsections.

Game	Result	Complexity	Technique
N/C	Recursive algorithm for memoryless strategy	Exponential $O(2^n)$	Attractors [2], [11]
N/C	Deterministic algorithm for memoryless strategy	Subexponential $n^{O(\sqrt{n})}$	Small paradises [6]
Min priority repeated is even	Deterministic algorithm for memoryless strategy	$O(dm.(\frac{n}{\lfloor d/2 \rfloor})^{\lfloor d/2 \rfloor})$ $O(dn)$ space	small progress measures - improvement [12]
N/C	Deterministic algorithm for memoryless strategy	sub-exponential $n^{O((n/d)^d)}$	discrete strategy improvement [14]
Weak parity (max priority reached is even)	Deterministic algorithm for memoryless strategy	Linear time	Attractors [18]
Generalised parity (boolean expressions of parity objectives)	memoryless strategy for disjunction finite memory strategy for conjunction	$n^{O(\sqrt{n})}.O(k.d).$ $(d_1, d_2, \ldots d_k)^d$	Attractors [20]

Table 3.1: Table of results for parity games

3.1 Parity games - algorithms for strategies

Consider a two player zero-sum game $\mathcal{G} = (V_0, V_1, E, \chi)$ where V_i is the vertex set of player i, E is the set of edges and $\chi : V \longrightarrow C$ is the priority function assigning priorities from C to vertices. Let $V = V_0 \cup V_1$, $\mid E \mid = m$, $\mid V \mid = n$ and $\mid C \mid = d$. The problem of *solving of a parity game* given a vertex $v \in V$, is to determine whether player 0 has a winning strategy for the play starting in v.

3.1.1 Exponential algorithm based on attractor method

McNaughton [4] proposed an elegant way of calculating the set of vertices from which the given set can be reached by a player in 0 or more moves. We have studied this under the name of Attractor in Section 2.2.1 earlier. This was adopted to parity games by Zielonka in [11] ([2]). However, using the attractor technique to crudely calculate the winning set for 0 yields exponential complexity as shown by the algorithm 3.1.

Algorithm 3.1 computes the winning sets of both players recursively. First it finds the set A of vertices with the maximum priority. Without loss of generality, let the parity of the maximum priority be even. It then solves the subgame $\mathcal{G}'$ induced by the set $V \setminus Attr_0(\mathcal{G}, A)$. If W_1' returned for the subgame is empty then 1 wins from all of vertices of $\mathcal{G}$. Else, W_1' is a part of 1's winning set (can be proved by a simple argument involving win sets as attractors) and the subgame $\mathcal{G}''$ induced by $V \setminus Attr_1(\mathcal{G}, W_1')$ is solved. The final solutions are given by the win sets of $\mathcal{G}''$.

Figure 3.1: Exponential algorithm for solving parity games based on attractors [2], [11]

Algorithm : Win
INPUT : Game $\mathcal{G} = (V_0, V_1, E, \chi)$
OUTPUT : Winning sets W_0, W_1

1. if $V = \emptyset$ return $(\emptyset, \emptyset)$
2. $d =$ maximum priority in $\mathcal{G}$
3. $A =$ set of vertices with maximum priority
4. $i = d \ mod \ 2$; $j = 1 - i$
5. $(W_0', W_1') = win(\mathcal{G} \setminus Attr_i(\mathcal{G}, A))$
6. if $W_j' = \emptyset$ then $(W_i, W_j) = (V, \emptyset)$
7. else $(W_0'', W_1'') = win(\mathcal{G} \setminus Attr_j(\mathcal{G}, W_j'))$ and
   $(W_i, W_j) = (W_i'', V \setminus W_i'')$
8. return (W_0, W_1)

This algorithm has been presented in [6] to compare with their algorithm.

Complexity

Let $T(n)$ be the time taken to solve a game of n vertices. The algorithm 3.1 makes two recursive calls with subgames each of which might have atmost $n-1$ vertices. Additionally, it preforms attractor computations which cost around $O(n^2)$. Hence $T(n) \leq 2T(n-1) + O(n^2)$. Solving this we obtain $T(n) = O(2^n)$.

3.1.2 Discrete strategy improvement algorithm

We shall now look into an algorithm for solving a parity game by improving a randomly chosen initial strategy until we can improve it no further. A detailed description can be found in [14]. The technique of strategy improvement for solving games was previously used for stochastic games [15] and discounted payoff and parity games [17]. However, due to the nature of the games these algorithms were mathematically involved and masked the beauty of the technique. Authors of [14] adopt this technique for simple parity games and with the help of additional information maintained for each vertex, solve the parity game without need for any involved calculations. The interesting aspect of this algorithm is information maintained for a vertex which has important details about the winning play involving this vertex.

Notations used in the algorithm

Note that the arena is considered to be bipartite here.

- $V_e \subseteq V$ ($V_o \subseteq V$) is the set of vertices with even (odd) priorities.
- Relevance ordering : orders the vertices according to their priorities i.e; $u < v$ iff $\chi(u) < \chi(v)$.
- Reward ordering on vertices: orders the vertices according to their importance to 0 i.e; $u \prec v$ iff $u < v \wedge v \in V_e \vee v < u \wedge u \in V_o$.
- Reward ordering on sets: $P \prec Q$ iff the most relevant vertex v in their symmetric difference is in Q if $v \in V_e$ or in P if $v \in V_o$.
- Reward ordering wrt another vertex: $P \prec_w Q$ same as above but with the sets restricted to contain vertices which are more relevant than w and $P \sim_w Q$ is the equivalence with respect to $\prec_w$.
- Play Profile of a vertex : $\psi(v)$ is a triple (u, P, e) where

  1. u is the most relevant vertex visited infinitely often,
  2. P is the set of more relevant vertices visited before first visit u and
  3. e is the number of vertices before first visit of u.

  All these are with respect to a play π and ψ is called a profile-vertex valuation. We will focus on only those valuations which are in accordance with a play i.e; they obey simple conditions such as "the value of e reduces as we proceed along the play but before the first visit of u'' and so on.
- Reward ordering on profiles : describing which of the two profiles is better for 0. $(u, P, e) \prec (v, Q, f)$ iff

  1. $u \prec v$ or
  2. $(u = v)$ and $P \prec Q$ or
  3. $(u = v)$ and $(P = Q)$ and $v \in V_o$ and $e < f$ or
  4. $(u = v)$ and $(P = Q)$ and $v \in V_e$ and $e > f$

  Similarly $(u, P, e) \prec_w (v, Q, f)$ and $(u, P, e) \sim_w (v, Q, f)$ can be defined.
- Strategy σ of a valuation ψ : is the one that picks the successor of a V_0 vertex as suggested by ψ i.e; the profiles of these vertices should be correctly updated as the play progresses.
- Improving a valuation : ψ' is improved with respect to ψ if ψ' picks the most rewarding vertex as the successor each time i.e; improvise the reward for 0 whenever possible.

INPUT: Game $\mathcal{G} = (V_0, V_1, E, \chi)$
OUTPUT : Winning sets : W_0, W_1 and strategies σ_0, σ_1

1. for each $v \in V_0$ select $\sigma_0(v) \in V_1$ such that $(v, \sigma_0(v)) \in E$
2. $\mathcal{G}' = (V_0, V_1, E', \chi)$ - among all edges in E only those suggested by σ_0 are retained in E'
3. $\psi = valuation(\mathcal{G}')$ - calculate the valuation in game $\mathcal{G}'$ compatible with σ_0 and is optimal for 1
4. $\sigma_0' = \sigma_0$
5. for each $v \in V_0$ select $\sigma_0(v) \in V_1$ such that $\psi(\sigma_0(v))$ is the most rewarding profile (according to $\prec$) if not already so
6. repeat 2,3,4,5 until $\sigma_0 = \sigma_0'$
7. for each $v \in V_1$ select $\sigma_1(v) \in V_0$ such that $\psi(\sigma_1(v))$ has the least rewarding profile
8. *RECALL : first component of a profile is the most relevant vertex repeated infinitely often*
9. $W_0 = \{v \in V \mid \text{first component of } \psi(v) \in V_e \}$
10. $W_1 = \{v \in V \mid \text{first component of } \psi(v) \in V_o \}$
11. return $W_0, W_1, \sigma_0, \sigma_1$

Figure 3.2: Discrete strategy improvement for solving parity games

Complexity

To compute the time taken by the Algorithm 3.2 we need to know the number of strategy improvement steps (number of times the *repeat–until* loop is repeated) and the time for each such step. Each step takes $O(nm)$ as it considers for each vertex all possible outgoing edges. The number of steps is the number of times the strategy is improved. Now this question of how many steps has been unanswered despite the large number of strategy improvement algorithms [15], [17] and [16]. Authors of [14] show that there exist algorithms which terminate after $O((n/d)^d)$ strategy improvement steps and that each step takes only $n^{O(1)}$. However, the Algorithm 3.2 terminates after at most n steps as it improvises the successor for each vertex at most once.

3.1.3 Deterministic sub-exponential algorithm [6]

As we discussed earlier, reducing the time taken to compute a winning strategy for a parity game is the most sought after problem. We shall now see the fastest known algorithm to solve a given parity game. Prior to [6] the best known complexity i.e; sub-exponential was given by [8] which was randomised and based on similar algorithm for stochastic games [7] which was inspired by randomised simplex algorithms [9], [10]. Along with retaining the sub-exponential complexity, the algorithm proposed in [6] is deterministic which makes it the best known algorithm so far.

Dominion $D \subseteq V$ is a small winning attractor of size l for a player i computed in $O(n^l)$ time (Lemma 4.1 [6]). Formally, D is said to be an $i-$dominion if i can win from every vertex of D without ever leaving D. It is computed as shown in module *dominion* of algorithm 3.3.

Jurziński et al propose an algorithm 3.3 in [6] which improvises on the exponential algorithm studied in section 3.1.1. The authors have reduced the complexity substantially (Algorithm 3.3) by calculating smaller attractor sets and avoiding redundant recalculations [6]. As is evident, the new algorithm calls the old algorithm whenever it is unable to find a *dominion* and if it does find one then it removes the attractor of the dominion and recurses on the left over game. The old algorithm is the same as algorithm 3.1 except that the recursive call is made to the new one.

Complexity

The dominions calculated are of size $l = \lceil\sqrt{2n}\rceil$. The recurrence relation for the time taken to solve a game of n vertices is

$$T(n) \leq O(n^l) + T(n-1) + T(n-l)$$

where in $T(n-1)$ is due to the first call to new-win in old-win and $T(n-l)$ is for the second new-win call in old-win. The second call had atmost $n-l$ vertices as the W'_j calculated in old-win is a $j-$dominion and hence of size atleast l. Solving this recurrence

Figure 3.3: Deterministic sub-exponential algorithm for solving parity games

INPUT : Game $\mathcal{G} = (V_0, V_1, E, \chi)$
OUTPUT : Winning sets W_0, W_1
Main Module : new-win($\mathcal{G} = (V_0, V_1, E, \chi)$)

1. d = maximum priority, n = number of vertices
2. $l = \lceil \sqrt{2n} \rceil$ *size of the dominion*
3. if $V = \emptyset$ return $(\emptyset, \emptyset)$
4. $(D, i) = dominion(\mathcal{G}, l)$
5. $j = 1 - i$
6. if $D \neq \emptyset$
   $(W_0', W_1') = new\_win(\mathcal{G} \setminus Attr_i(D))$
   $(W_j, W_i) = (W_j', V \setminus W_j')$
7. else $(W_0, W_1) = old\_win(G)$
8. return (W_0, W_1)

Module : old-win($\mathcal{G} = (V_0, V_1, E, \chi)$)

1. A = set of vertices of highest priority
2. $i = dmod2$
3. $j = 1 - i$
4. $(W_0', W_1') = new\_win(G \setminus Attr_i(A))$
5. if $W_j' = \emptyset$ then $(W_i, W_j) = (V, \emptyset)$
6. else $(W_0'', W_1'') = new - win(G \setminus Attr_j(W_j'))$ and
   $(W_i, W_j) = (W_i'', V \setminus W_i'')$
7. return (W_0, W_1)

Module : dominion$(\mathcal{G}, l)$ returns $i-$dominion of size l and player i

1. Compute all subsets of size l
2. for each subset U of size l do all the following
3. check if it is a 0-trap or 1-trap
4. if both fail then U is not a dominion
5. if U is a $i-$trap then Subgame $\mathcal{G}' = \mathcal{G}[U]$ and
   if $1 - i$ wins from all states of $\mathcal{G}'$ then return U as $(1 - i)$-dominion
   use Algorithm 3.1
6. else U is not a dominion; try next subset

gives total runtime to be $n^{O(\sqrt{n})}$.

Consider a game with arena which has atmost two outgoing edges for every vertex. The authors show that the running time for such a game is $n^{O(\sqrt{n/log\ n})}$ by choosing $l = \lceil \sqrt{n/log\ n} \rceil$.

3.1.4 Small progress measures for solving parity games [12]

Section 3.1.2 achieves the best known complexity maintaining additional information for each vertex. We shall now look into an algorithm [12] which also maintains additional information for each vertex. The information for a vertex acts as a witness for the winning strategy indicating which direction to proceed to reach a cycle with the smallest priority being even. This can be easily modified for maximum priority being even.

Close observation of the parity objective tells us that a play is winning for 0 iff it eventually reaches a cycle whose smallest priority is even. We call such cycles as *even cycles* and try to indicate for each vertex whether it belongs to an even cycle.

Notation used in the algorithm

Recall that d is the number of priorities in the game $\mathcal{G}$.

1. Tuple ordering : $(a_1, a_2, a_3 \cdots a_d) \sim_i (b_1, b_2, b_3 \cdots b_d)$ denotes the lexicographic ordering $\sim$ between the two $d-$tuples applied to the first i components.

2. **Progress measure** : A function $\varrho : V \rightarrow \mathbf{N}^d$ is a parity progress measure if $\forall (v, w) \in E$, $\varrho(v) \geq_{\chi(v)} \varrho(w)$ and the inequality is strict if $\chi(v)$ is odd. Intuitively, a progress measure indicates the existence of only even cycles.
   The lexicographic ordering for tuples is extended to progress measures such that $\varrho \sqsubseteq \mu$ if forall v $\varrho(v) \leq \mu(v)$.

3. M_G : is the set of $d-$tuples such that each tuple has only zeros on even positions and odd positions i have a positive integer in $\{1, 2, \cdots |V^i|\}$ where V^i is the set of vertices of priority i. By $M_G^\top$ we denote the set $M_G \cup \{\top\}$ where $\top$ is the biggest element.

4. $prog(\varrho, v, v)$ for a parity measure $\varrho : V \rightarrow M_G^\top$ and $(v, w) \in E$ is the least $m \in M_G^\top$ such that $m \geq_{\chi(v)} \varrho(w)$ and if $\chi(v)$ is odd then the inequality is strict or is the top element. Intuitively, $prog(\varrho, v, w)$ picks for an edge (v, w) the same value as indicated by $\varrho(w)$ or the next value if $\chi(v)$ is odd and the comparison is up to the first $\chi(v)$ positions.

5. **Game parity progress measure** : a progress measure ϱ is called a game parity progress measure if

   - for all $v \in V_0$, $\varrho(v) \geq_{\chi(v)} prog(\varrho, v, w)$ for some $(v, w) \in E$ and
   - for all $v \in V_1$, $\varrho(v) \geq_{\chi(v)} prog(\varrho, v, w)$ for all $(v, w) \in E$

Figure 3.4: Algorithm : computing the winning game parity progress measure
INPUT : Game $\mathcal{G} = (V_0, V_1, E, \chi)$
OUTPUT : Winning game parity progress measure ϱ

1. Let ϱ be a measure which assigns the tuple $(0, 0, \cdots 0)$ to all vertices
2. while $\varrho \sqsubset Lift(\varrho, v)$ for some $v \in V$
3. do $\varrho = Lift(\varrho, v)$
4. return ϱ

Intuitively, a game parity progress measure is the one that indicates the winning strategy's moves for V_0 by indicating the successor and ensures that 1 cannot ruin the play as it requires all successors of a V_1 vertex to adhere to the tuple ordering. A value $\top$ assigned to a vertex indicates that this vertex is losing for 0.

6. **Strategy** σ_0 **for** ϱ : picks the successor having the least value of ϱ.
7. $Lift(\varrho, v)(u)$ is given as follows
   - $\varrho(u)$ if $u \neq v$
   - $max\{\varrho(v), min_{(v,w)\in E} prog(\varrho, v, w)\}$ if $u = v \in V_0$
   - $max\{\varrho(v), max_{(v,w)\in E} prog(\varrho, v, w)\}$ if $u = v \in V_1$

   Intuitively, $Lift(\varrho, v)$ improves the value assigned to v according to $prog$ such that $\varrho(v)$ is retained if $v \in V_0$, is updated to the least value larger than $\varrho(v)$ (up to $\chi(v)$ positions). It improves ϱ for a given vertex so that the value is now optimal for 0 i.e; indicates the winning even cycle.

Algorithm

From the definition of $Lift$ operator given at 7 in the list above, it is clear that a progress measure that has been lifted for all vertices is our final progress measure which gives the winning strategy. Hence the simultaneous fix-point of $Lift(\varrho, v)$ for all v will be our winning measure.

Complexity

The algorithm 3.4 takes $O(d.n)$ space as it keeps just a $d-$tuple for each of the n vertices. Its running time is $O(d.m.(\frac{n}{\lfloor d/2 \rfloor})^{\lfloor d/2 \rfloor})$. We omit the details here. The authors claim that their algorithm can be modified to have a $O(d.m.(\frac{n+d}{d})^{\lceil d/2 \rceil})$ as its worst-case running time.

Optimizations : The running time can be improved by pre-computing values of the least progress measure, decomposing the game into maximal strongly connected components as we work with cycles.

Improvement of the algorithm [13]

The algorithm using progress measures for each vertex of the game studied above determine the winning set for player 0 by assigning an integral value $\neq \top$ while all those states whose measures are $\top$ are winning for player 1. [13] analyses the exact role played by the measure in the winning set of 0 given that the priorities of the vertices are 0,1 and 2. The authors conclude that the measure assigned to a winning vertex is always below the number of states with priority 1 (denoted by V^1). As the algorithm studied above works to find a fix-point iterating each time it can improve the measure of atleast one vertex, the number of iterations it would need are proportional to V^1.

[13] proposes a faster way of reaching the fix-point while maintaining the worst-case behavior by reducing the number of iterations whenever possible. Let there exist a measure $0 < k < V^1$ which is not assigned to any vertex but there are vertices with measures $> k$. Such a measure is called a *k-gap* and the authors prove that any vertex with a measure larger than k is losing for 0 and is given the measure $\top$. Such an improvement reduces the number of iterations needed to reach a fix-point. However, it is not clear whether how well this trick extends to games with more than 3 priorities.

3.2 Modified Parity Games

There are several variations of parity games each of which attempts to reduce the complexity of solving the game. Some variations restrict the number of priorities, some restrict the number of out-edges, some restrict the winning condition itself. On the other hand, due the growing popularity of parity games, several research directions opened up experimenting with the parity conditions by generalizing them, adopting existing parity techniques, and comparing the expressiveness and determinacy with those of parity games.

We shall consider in the coming subsections two modifications in opposite directions with one restricting the parity condition and the other generalizing it. As one would expect, the complexity for the weak condition is much lower than that known for parity games while that for the generalized condition is much higher requiring memory for winning strategies in some cases.

3.2.1 Weak Parity Game

A *weak parity game* is a normal parity game with the modification that the winning objective for player 0 consists of all infinite plays whose least priority visited is even. This class of games is well suited for solving reachability games with an additional requirement on the priorities. This objective was first considered in [19] which solved the game and showed the run time to be $O(d, m)$[1]. However, a keener observation and improved analysis of the algorithm show the run time to be $O(m)$ [18].

[1]Recall that d is the number of priorities, m is the number of edges and n is the number of vertices in the game

Figure 3.5: Algorithm to solve weak parity games

1. **INPUT** : Game $\mathcal{G} = (V_0, V_1, E, \chi)$
2. **OUTPUT** : Winning sets (W_0, W_1) that partition V
3. $\mathcal{G}^0 = \mathcal{G}$; $W_0 = W_1 = \emptyset$
4. For each priority j,
   - Let $i = j \ mod \ 2$
   - find the attractor A_j for the set of vertices with priority j for player i
   - This attractor is part of the W_i
   - $\mathcal{G}^{j+1} = \mathcal{G}^j \setminus A_j$

   End for
5. return (W_0, W_1)

The algorithm is based on the calculation of attractors presented in Section 2.2.1. It works as follows.

It works by calculating the attractors for each priority and working on the subgame removing these attractors. The removed attractors form parts of winning sets depending on the priority they correspond to. Since it works by removing vertices, we believe that it terminates.

Complexity

A naive analysis of this gives the run time to be $O(d.m)$ as it works on the graph with d priorities and for each priority, we work on the edges to find the calculate the attractors. However, [18] cleverly shows that several edges are removed each time we shrink the game. The costliest operation performed in this algorithm is attractor computation. We know that for the computation of an attractor we analyze edges of a graph. Thus the total number of edges actually analyzed throughout the entire game is just m and hence the run time is just $O(m)$. Additionally, [18] provides efficient ways to compute attractors and the set of vertices of a priority in a shrunk game.

3.2.2 Generalized Parity Games [20]

Simple parity condition studied in the Sections 3.1 are sufficient to deal with all ω-regular objectives. However, there are other important winning conditions namely Rabin and Streett which cannot be captured by simple parity conditions. [2] shows that a Streett condition can be expressed as a Rabin condition. Simple parity conditions (NP $\cap$ Co-NP [23]) are actually special cases (closed under complementation) of both Rabin (NP-

Figure 3.6: Algorithm to solve generalized parity games

INPUT : Game $\mathcal{G} = (V_0, V_1, E, \{\chi_1, \chi_2, \cdots \chi_k\})$ [Player 0 has conjunction and 1 has disjunction of parity objectives]

OUTPUT : Winning sets W_0, W_1

1. Pick one of the disjuncts and let its max even priority be d
2. The set of states from which 1 wins by (a) visiting d finitely often or (b) visiting $d-1$ infinitely often or (c) win by lower priorities or (d) win by other disjuncts
3. Find the attractor A for 1 for priority $d-1$. This is a trap for 0 where she cant win.
4. Additionally find an attractor B for 1 for a simpler winning condition i.e; subgame without the set A
5. Both A and B are winning for 1. So we remove these two and proceed with the subgame in a recursive fashion.
6. After this has been done for all disjuncts, the remaining states are winning for 0.

complete) and Street (Co-NP complete) [22].

By the nature of the conditions (studied in Chapter 2) it is clear that each pair in a Rabin family can be captured by a simple parity condition. Thus to capture the entire winning objective posed by Rabin condition, a disjunction of simple parity conditions. Similarly Streett can be expressed as a conjunction of parity conditions.

Rabin-Street conditions are duals of each other and hence fit well together as complementary objectives of the two players in a zero-sum game. Rabin games enjoy memoryless determinacy while Street have finite memory determinacy [21]. The known complexity to solve Rabin and Street games was $O(m, n^k, k!)$ [21] where k is the number of pairs in the family. [20] retains these determinacies and improves the complexity to $n^{O(\sqrt{n})}.O(k.d).(d_1, d_2 \overset{d}{\cdots} d_k)$ where $d_1, d_2 \cdots d_k$ are priorities for the k pairs.

The classical algorithm for conjunction and disjunction was given by Zielonka. We shall briefly describe that algorithm for disjunction now. Let $d_1, d_2 \cdots d_k$ are the k priorities assigned to a vertex. As they are duals of each other we shall assign conjunctions to player 0 and disjunctions to player 1. Note that by disjunction or conjunction of parity objectives we mean that each objective is simple parity objective for the player. For example the disjunction for player 1 means that a play is winning for 1 if the least priority repeated infinitely often is odd for atleast one of the k priorities. The actual priority objectives for player 1 considered in [20] are obtained by complementing the priority functions given and requiring the maximum priority to be even. But for the sake of simplicity we have taken the same functions and require the max priority to be odd for 1. Intuitively, this algorithm keeps multiple simple parity condition solving algorithms active at the same

time. It finds the winning sets and their attractors for each of the simple conditions. These are then removed and it recurses on the left-over game.

Complexity

[20] improve the complexity by calculating the attractors efficiently. To do this, they use the concept of *dominions* introduced in [6]. They identify winning sets in the subgames as dominions. As these are small winning attractors with efficient algorithms to compute them, the algorithm can use this to calculate attractors and reduce complexity to $n^{O(\sqrt{n})}.O(k.d).(d_1, d_2^{d} \cdots d_k)$.

It is easy to see that due to the rank-based memoryless strategy for attractors described in section 2.2.1, player 1 has memoryless determinacy. As player 0 has a conjunction of objectives to satisfy, she needs to remember all of them and hence needs atleast finite memory for her winning strategy.

Chapter 4

Timed Games

In this chapter we will focus on games played on an arena with timing information. Traditionally, this arena is a timed automaton [1] with some slight deviations. We shall first give a brief notational introduction to timed automata and the main result of language emptiness decidability.

Essentially, timed automata are finite automata augmented with clocks which grow at the same rate and maintain timing information. These are variables over which we can write boolean expressions to check their values and assignment expressions to update their values. As the clocks take real values, the number of configurations (states) in a timed automaton are infinite. Thus it is difficult to use the algorithms known for finite automata to check for language emptiness.

The decidability of language emptiness is due to the existence of an equivalence relation of finite index on the set of all states. Once this relation has been proved to exist, a finite automata (untimed) can be built whose language is symbol-equivalent to the timed language. (we will study the notion of this equivalence later.) However there are several generalizations [26], [27] of the classical timed automata which do not enjoy this decidability. Even more interesting aspect of this relation is that it exists for a particular generalization and suddenly becomes unavailable for slight variations of the model especially if we are varying the number of clocks in the model. It has been a long standing open problem whether an equivalence relation exists for several models with 3 clocks [25], [27].

As it turns out, this concept of equivalence classes is central to all kinds of analysis questions posed on timed automata. Several timed games we will study in the coming sections of this chapter have positive results precisely because of this equivalence relation. The game structure given has a timed automaton for its arena. This arena is abstracted out into a finite automata using the finite index equivalence relation on the infinite state set. The timed game is then structured into an equivalent untimed one seeking a goal very similar to the timed game.

We shall first give the list of latest-most improved results for timed games, then we shall study in the Section 4.2 the class of timed games.

Let X be the set of clocks, c_m the maximum constant appearing the clock constraints,$|L|$ be the number of locations, $S_{\mathcal{R}}$ be the set of states (equivalence classes) in the region automaton with $\mathcal{R}$ as the set of regions. As in the untimed games, turn-based timed games partition the set of locations while concurrent games partition the set of actions. Let L_i (E_i) be the set of locations (actions) available to player i. The table 4.1 gives the scenario for timed games. Unless specified otherwise, the table deals with two player zero-sum games with timed automata as arena. The plays are infinite but need not be turn based. We shall mention for each game the deviations from this description and the winning conditions. H stands for the set of locations in the parity automaton representing the goal.

4.1 Preliminaries of timed automata

We consider as time domain $\mathbb{T}$ the set $\mathbb{Q}_+$ or $\mathbb{R}_+$ of non-negative rationals or reals, and A a set of actions. A time sequence over $\mathbb{T}$ is a non-decreasing sequence $\tau = (t_i)_{i\geq 1}$; for simplicity t_0 is taken to be zero always. Let X be a set of clocks. A clock valuation over X is a mapping $\nu : X \to \mathbb{R}_+$. We denote by $\mathbb{R}_+^X$ (or $\mathbb{T}^X$) the set of clock valuations over X. If $\nu \in \mathbb{T}^X$ and $\tau \in \mathbb{T}$, then $\nu+\tau$ is the clock valuation defined by $(\nu+\tau)(x) = \nu(x)+\tau$, for $x \in X$. A guard or constraint over X is a conjunction of expressions of the form $x \sim c$ where $x \in X$, $c \in \mathbb{N}$ and $\sim \in \{<, \leq, >, \geq, =\}$. We denote by $C(X)$ the set of guards over X. The satisfaction relation for guards over clock valuations is denoted as $\nu \models \varphi$ whenever valuation ν satisfies guard φ in the usual way.

Clock constraints allow us to test the values of clocks. To change the value of a clock x we use clock resets. $U_0(X)$ denotes the set of resets $\phi \in U_0(X)$ defined as $\phi \subseteq X$.

Let ν be a valuation and $\phi \subseteq X$ be a clock reset. We use the notation $\nu' = \nu[\phi := 0]$ to denote $\nu'(z) = \nu(z)$ for all $z \in X \backslash \phi$ and $\nu'(y) = 0$ for all $y \in \phi$.

4.1.1 Timed Automata

A *timed automaton* [1] is a tuple $\mathcal{A} = (L, L_0, A, X, E, F)$ where L is a set of locations; $L_0 \subseteq L$ is a set of initial locations; A is a set of symbols; X is a set of clocks; $E \subseteq L \times L \times A \times C(X) \times U_0(X)$ is the set of edges and $F \subseteq L$ is a set of final locations. $C(X)$ and $U_0(X)$ are the set of clock constraints and clock resets as described above. An edge $e = (l, l', a, \varphi, \phi)$ represents an edge from l to l' on symbol a, with the valuation $\nu \in \mathbb{T}^X$ satisfying the guard φ, and then ϕ gives the resets of certain clocks.

Timed Transition System

It is clear that due to infinitely many values taken by clocks, the number of states (l, ν), $l \in L, \nu \in \mathbb{T}^X$ is infinite for a timed automaton. We define for a given timed automaton $\mathcal{A}$ the timed transition system $T_{\mathcal{A}}$ over the set of states $L \times \mathbb{T}^X$. The state of a timed automaton can change in 2 ways:

Game	**Result**	**Complexity**	**Technique**
Concurrent Parity	memoryless - safety no memoryless -reach	$O(\|L\|.\|X\|!.2^{\|X\|}.(2C_{max}+1)^{\|X\|}.\|H\|.\|H_*\|^{\|H\|+1})$	Symbolic algorithms [33] Goal parity automaton
Concurrent Parity	memoryless - safety finite memory -reach	EXPTIME complete	Randomised time strategy [34] parity as reach and safety
Concurrent Parity	Reduction to turn based untimed parity	Edges = $\|S_{\mathcal{R}}\|.((c_m+1).(\|X\|+2).2).$ $(\|E_1\|+1).[(\|E_2\|+1).((c_m+1).(\|X\|+2).2)]$ States = $\|S_{\mathcal{R}}\|.(1+(c_m+1.(\|X\|+2).2.(E_1+1)))$	Regions to untime [29] apply actions sequentially
Concurrent Time to reach	Receptive strategy	$O((\|L\|.(c_m+1)^{\|C\|}\|C+1\|!2\|C\|)^2\|C\|)$ EXPTIME complete	Fix point μ expression [31] on extended clock regions
Turn Time to reach	Functions to compute time, # to reach	$O(\|\mathcal{R}\|^3)$ EXPTIME complete for 2 clocks	Strategy improvement [30], of regionally constant strategy
Turn Min/Max avg time/edge	Reduction to avg-price untimed	EXPTIME complete for 2 clocks	Regions consistent [32] with values

Table 4.1: Table of results for timed games

1. *Due to elapse of time*: for a state (l, ν) and a real-number $t \geq 0, (l, \nu) \xrightarrow{t} (l, \nu + t)$. This kind of transition is called a *timed transition*.

2. *Due to a location-switch*: for a state (l, ν) and an edge $(l, l', a, \varphi, \phi)$ such that $\nu \models \varphi$, $(l, \nu) \xrightarrow{a} (l', \phi(\nu))$. We call such a transition, an A-transition.

A path is a finite (infinite) sequence of consecutive (time or location-switch) transitions. The path is said to be accepting if it starts in an initial location ($l_0 \in L_0$) and ends in a final location (or repeats a final location infinitely often). A run through a path from a valuation ν_0' (with $\nu_0'(x) = 0$ for all x) is a sequence $\pi = (l_0, \nu_0') \xrightarrow{t_1} (l_0, \nu_1) \xrightarrow{(a_1, \varphi_1, \phi_1)} (l_1, \nu_1') \xrightarrow{t_2} (l_1, \nu_2)$ The timed word corresponding to the run is $tw = (a_1, t_1)(a_2, t_2)(a_3, t_3) \cdots (a_n, t_n)$. Note that $\nu_i = \nu_{i-1}' + (t_i - t_{i-1}), \nu_i \models \varphi_i$, and that $\nu_i' = \nu_i[\phi_i := 0], i \geq 1$. A timed word tw is accepted by $\mathcal{A}$ iff there exists an accepting run (through an accepting path) over $\mathcal{A}$, the word corresponding to which is tw. The timed language $L(\mathcal{A})$ accepted by $\mathcal{A}$ is defined as the set of all timed words accepted by $\mathcal{A}$.

Note that we refer to $l \in L$ as a location and $e \in E$ as an edge while (l, ν) as a state and $(l, \nu) \longrightarrow (l', \nu')$ as a transition.

4.1.2 Region Automata

Given a set X of clocks, let $\mathcal{R}$ be a partitioning of $\mathbb{T}^X$. Each partition contains a set (possibly infinite) of clock valuations. Given $\alpha \in \mathcal{R}$, the successors of α represented by $Succ(\alpha)$ are defined as

$$\alpha' \in Succ(\alpha) \text{ if } \exists \nu \in \alpha, \exists t \in \mathbb{T} \text{ such that } \nu + t \in \alpha'$$

The partition $\mathcal{R}$ is said to be a *set of regions* iff

$$\alpha' \in Succ(\alpha) \Longleftrightarrow \forall \nu \in \alpha, \exists t \in \mathbb{T} \text{ such that } \nu + t \in \alpha'.$$

A set of regions is consistent with time elapse if two valuations which are equivalent (within the same partition) stay equivalent with time elapse. A region $\alpha \in \mathcal{R}$ is said to satisfy a clock constraint $\varphi \in C(X)$ denoted as $\alpha \models \varphi$, if $\forall\ \nu \in \alpha, \nu \models \varphi$. A clock reset $\phi \in U_0(X)$ maps a region α to a region $\alpha[\phi := 0] = \alpha'$ such that $\alpha' \cap \{\nu[\phi := 0]\} \neq \emptyset$ for $\nu \in \alpha$.

A set of regions $\mathcal{R}$ is said to be *compatible* with a set of clock constraints $C(X)$ iff $\forall \varphi \in C(X)$ and $\forall \alpha \in \mathcal{R}$ exactly one of the following holds (a) $\alpha \models \varphi$ or (b) $\alpha \models \neg\varphi$. A set of regions $\mathcal{R}$ is said to be *compatible* with a set of clock resets $U_0(X)$ iff $\alpha' = \alpha[\phi := 0] \Rightarrow \forall \nu \in \alpha, \exists \nu' \in \alpha'$ such that $\nu' = \nu[\phi := 0]$.

Given a timed automaton $\mathcal{A}$, and a set of regions $\mathcal{R}$ compatible with $C(X)$ and $U_0(X)$, the *region automaton* $\mathcal{R}(\mathcal{A}) = (Q, Q_0, A, E', F')$ is defined as follows: $Q = L \times \mathcal{R}$ the set of locations; $Q_0 = L_0 \times \{\alpha_0\}$ (α_0 is the region where $\nu(x) = 0$ for all $x \in X$), the set of initial locations; $F' = F \times \mathcal{R} \subseteq Q$ the set of final locations; $E' \subseteq (Q \times A \times Q)$ is the set of edges. $(l, \alpha) \xrightarrow{a} (l', \alpha')$ if $\exists \alpha'' \in \mathcal{R}$ and a transition $(l, l', a, \varphi, \phi) \in E$ such that (a) $\alpha'' \in Succ(\alpha)$, (b) $\alpha'' \models \varphi$ and (c) $\alpha' = \alpha''[\phi := 0]$.

Let $tw = (a_1, t_1)(a_2, t_2)(a_3, t_3) \cdots (a_n, t_n)$ be a timed word then $\text{Untime}(tw) = a_1 a_2 \cdots a_n$. Also, $\text{Untime}(L(\mathcal{A})) = \{\text{Untime}(tw_i) \mid \exists tw_i \in L(\mathcal{A})\}$. The region automaton is an abstraction of the timed automaton accepting $\text{Untime}(L(\mathcal{A}))$ [1].

Theorem 4.1. *Let $\mathcal{A}$ be a timed automaton. Then the problem of checking emptiness of $L(\mathcal{A})$ is decidable.*

4.2 Games played on timed automata

In this section we shall study the strategies available to a player in a timed game over timed automaton. [35], [37] paved the way for timed games. Treating timed automata as a game structure and building winning strategies for player 0 (controllers in these papers) was the beginning of timed games over timed automata. A series of papers [33], [29], [30], [31], [32], [34] followed which dealt with different winning objectives, adding meaning to a timed strategy, improving the result by introducing randomness and so on. However, most of the papers rely on the classic region equivalence available to a timed automata and reduce the given timed game to the corresponding untimed version.

The notions of game introduced in Chapter 2 can be extended in a simple fashion to timed games by replacing the arena of untimed graph with timed automata. The moves proposed or selected by the players in untimed games are outgoing edges from the vertex. As the infinite transition system of a timed automaton differs from the finite representation, the players actually carve out a timed path in the transition system by proposing timed moves (t, a) from a state (l, ν). While there is no notion of a language in a timed game, the goal or objective is represented as a set of desirable paths in the transition system. There are several ways of representing this objective with the simplest form being extending the untimed objectives to a timed setting by considering the objective to hold only with respect to the locations and hence called *location goals*. For example, if we consider a parity objective in a timed game, it would require the infinitely repeated priority to be even plays winning to player 0. This objective places no timing restriction and hence the objective is with respect to the location and not the clock valuation. However, there is some work on timed objectives too [36]. A strategy of a player tells for a given prefix of the play the move to be proposed.

4.2.1 Concurrent timed games

Though the notion of concurrency is not fully matured in a timed setting, several papers have attempted to capture this essential aspect in a game. [33] was the first attempt in this direction. Concurrency is expressed by requiring both players to propose moves of the form (t, a) where t is the time player wants to elapse before executing action a. Hence the edge set E is partitioned into E_1 and E_2 The move with smaller of the two delays is chosen and that player gets her way. The immediate concern is that of time divergence as the player how proposes 0 delay always gets to play her actions. Thus there is a trivial winning strategy for any player. But such a strategy is not physically meaningful as it induces zeno-plays and are thus invalid. Hence for a strategy to be physically meaningful it has to ensure *time divergence*.

Several attempts have been made to achieve time divergence, placing syntactic restrictions [37], [38], [36] or semantic restrictions [39]. Others [35] require the controller (player 0) to ensure time divergence. While all these means do achieve the goal they are in some sense too restrictive and limit the expressiveness. [33] followed by [29], [31] and [34] follow a rather decent way of capturing time divergence by imposing certain additional requirements on the winning objective. This is by far the least restrictive with respect to

the game's expressiveness.

Given a goal Φ for player i, the actual winning objective in [33] for which the winning sets and strategies are computed is given by $WC_i(\Phi)(\Phi \cap td) \cup (Blameless_i \setminus td)$ where td represents the set of time divergent runs (where in the total time elapsed over the infinite run tends to infinity) and $Blameless_i$ is the set of runs where i is responsible for only finitely many moves (and hence even if the run is converging, as $1 - i$ is guilty of this convergence, i still wins the play). Such a modification along with giving a meaningful expression to time divergence also ensures that for a given play, there is atmost one winner. However, it is possible that there are plays which are won by neither player and this also results in some locations of the automaton being winning for neither of the players. A simple proof of this lack of run-level determinacy can be given by considering a run where both players propose moves 0 delays throughout the game. Such a game cannot be declared winning for either player as it would be against the notion we are trying so hard to capture. This is a serious drawback in timed games leading to the following theorem. Note that in such a game we can strive for two levels of determinacy namely *strong* and *weak*. A game is strongly determined if plays losing to 0 can be won by 1 and a game is weakly determined if 1 can force the play (losing to 0) away from the winning plays of 0 i;e 1 can force the game into $WC_1(\sim \Phi)$ or $\sim WC_0(\Phi)$ respectively.

Theorem 4.2. *The timed automaton games (and hence, the timed game structures) are neither weakly, not strongly determined for the class of reachability goals. [33].*

It is also shown in [33] that while safety objective can have memoryless strategies in both timed and untimed games, reachability has memoryless in untimed and only finite memory strategies in timed games.

Goal as a parity automaton [33]

[33] and [34] deal with timed games by using the elegant techniques proposed in [40] for untimed games with an infinite state set. They achieve this by first expressing the given $\omega-$regular objective as a parity objective. This objective is then translated into a deterministic and total parity automaton which is a deterministic finite automaton with labels and priorities derived from the locations of the timed automaton. The totality and determinism are with respect to the labeling function which assigns to each vertex in the parity automaton the name of a location in the timed automaton.

Time divergence is imposed by extending the states with additional information such as *tick* which is true if the clock valuation of the state's predecessor saw an integer unit elapse, *blame* is true if 0 was responsible for the preceding transition. It is easy to see that using these additional information, the set of meaningful winning plays i.e; the set $WC_i(\Phi)$ for player i can be captured. The states of both the timed automaton and the parity automaton are augmented with information *tick* and *blame*. A $\mu-$calculus formula is derived for the winning plays by finding the time divergent fix point of the *Controlled predecessor* operator. Controlled predecessor of a state indicates the set of states from which 0 can force a visit to the state in one move.

The solution for timed games outlined in [33] is along the same lines as [40] but for the timing part requires all the $\mu-$calculus formulae over the parity automata to be regionally constant (i.e; for any two states with their locations being the same and their clock valuations being in the same region the operators in the formula behave alike). The strategy in a timed game proposes the move (t, a) from a state (l, ν) such that $(l, \nu) \xrightarrow{t} (l, \nu + t) \xrightarrow{a} (l', \nu')$ where l' would have been chosen by the strategy for untimed version in [40] and t is chosen such that the action a is enabled.

Complexity -The time taken to compute the winning set is $O(|L|.|X|!.2^{|X|}.(2C_{max} + 1)^{|X|}.|H|.|H_*|^{|H|+1})$ where H is the set of states in the parity automaton and H_* is the set of states in the parity automaton augmented with additional information. The rest of the variables pertain to the timed automaton (described in Section 4.1). [33] also proves that solving a timed game for a location goal is EXPTIME complete.

Trading memory for randomness [34]

The strategy outlined above from [33] proposes a time delay t for every move by accessing an infinitely precise clock and hence requires infinite memory to hold the value of this clock. [34] proposes to use only finite memory for the strategies but at the cost of randomizing them. Note that all the goals are location goals.

Here too the given timed automaton states are augmented to impose time divergence. Winning states and strategies for set B of safety states are achieved by solving the untimed version of the game over the region automaton for reachability of $L \setminus B$ and reverting the winning sets for 0 and 1. The strategy for 1 is the rank based strategy for attractor set $Attr_1(\mathcal{G}, L \setminus B)$ while 0 chooses any outgoing edge in the remaining states which form a trap for 1. All the states are augmented with additional information to help ensure time divergence by requiring at least a few clocks to be reset infinitely often. The strategy can propose any time delay because no matter what 1 does it cannot escape the winning set of 0 which is a trap to 1.

For solving the reachability objective, the attractor to the set of states is computed and the rank based strategy is used to predict the action part of the move (t, a) from the state (l, ν). Let $(l, \nu) \xrightarrow{t} (l, \nu + t) \xrightarrow{a} (l'\nu')$. The time delay t is predicted by taking the least possible time taken to reach the region containing the state $(l, \nu + t)$. If such a minimum does not exist i.e; the interval of times to reach the region is not left closed then the strategy randomly and uniformly chooses a value within the interval. As it does not pick an exact value to wait, the need for a precise clock is eliminated. However, such strategies can be implemented only for those controllers which can wait a random amount of time. As they do not require to know the precise time delay they do not need a precise clock and an imprecise one would do as well.

The authors outline a very simple algorithm for solving timed parity games too. Let us first understand how to solve an untimed parity game through reachability and safety.

- Let $\mathcal{G}^0 = \mathcal{G}$ be the given game and $i = 0$.

- Find a set of states $Attr_0(\mathcal{G}^i, V^d)$ of the game $\mathcal{G}^i$ for the vertices of maximum even priority d.
- Solve the left over subgame $\mathcal{G}^{i'} = \mathcal{G}^i[V \setminus Attr_0(\mathcal{G}, V^d)]$ with less number of priorities to obtain the winning set Y^i for 0.
- Solve the game $\mathcal{G}^i$ for the safety objective of staying within Y^i to obtain the winning set Z^i.
- The same is repeated for the subgame induced by $V^i = V \setminus Z^i$ until $V^i = Z^i$. This V^i is winning for 0.

Note that Controlled predecessor operator is used in the construction of *Attr* sets.

Intuitively, the notion of winning set in a parity game can be expressed as those states from which we can reach the states with desired priorities (reachability requirement) and once we have reached these states, the play can stay there forever (safety requirement). The strategy from the winning states is built by using the strategies of reachability and safety in the corresponding sets. Strategy for reachability objective in the states in $Attr_0(\mathcal{G}, V^d)$ the attractor rank based strategy is used and the strategy for safety objective in the set Z^i is by arbitrarily choosing some state in the set Z^i (works by the definition of winning set for a safety objective). The timed version is similarly solved for winning sets and strategies.

Memory requirements- The authors show that reachability strategies and thus parity strategies can not be memoryless. If they are memoryless then 1 can easily figure out the least delay proposed by 0 and always propose lower than that but beyond 0 time units and always win the play. Thus to ensure that 1 does not collect all the delays proposed by 0 and propose lower than the least of 0's delays, 0 has to propose infinitely many delays and to do so 0 would need a precise clock. However, they are not sure if safety can be achieved by memoryless strategies. It is important to note that due to the use of an imprecise clock for randomised strategies only finite memory is needed. Thus for safety objective deterministic finite memory strategies exist while for reachability and parity randomised finite memory or deterministic infinite memory (outlined in the section 4.2.1) exist.

Untiming the timed game [29]

The techniques of solving timed games that we saw above though elegant do not exploit the the main feature of a timed automaton. Most results pertaining to timed automata rely largely on the classical region equivalence. Now we shall brief a technique which untimes the given timed parity game [29].

The notion of time divergence and the means of achieving it by modifying the winning objective is adopted here too. The states of the timed automata are augmented with the boolean flags *tick* and *blame* which carry the same meaning as earlier. Now a region automaton is constructed considering regions over these augmented states. However, as

the players concurrently their actions are disjoint and hence the set E in the timed automaton is partitioned.

In the timed game, from a state (l, ν) player i had the move (t_i, a_i) such that i reaches $(l, \nu + t_i)$ and then (l^i, ν^i) if her move was picked. In the region automaton, from a state $s_r = [(l, \nu)]$ player 0 always picks the region R_0 and action a_0 indicating that she would like to elapse enough time to reach R_0 and then take the action a_0. After 0 has reached her desired state $s_r^0 = [(l^0, \nu^0)]$ in the region automaton, 1 proposes her own region, action pair R_1, a_1 to reach $s_r^1 = [(l^1, \nu^1)]$ provided she move in the timed game could defeat 0's move. Thus the given timed parity game is converted into an untimed finite bipartite turn based parity game.

The winning states for 0 in the region automaton give the locations winning for her in the original timed game. The proof of this follows immediately from the proof of language emptiness decidability in [1]. Now, any of the techniques described in Chapter 3 can be used to solve the untime parity game.

Complexity-As the solution is largely based on the conversion to an untimed automaton, the main contributing factor to the run time is the size of the region automaton. The number of states in the region automaton is $|S_{\mathcal{R}}|.(1 + (c_m + 1)|.(|X| + 2).2.(|E_1| + 1))$ and the number of edges is $|S_{\mathcal{R}}|.((c_m + 1).(|X| + 2).2).(|E_1|)[1 + (E_2 + 1).((c_1).(|X| + 2).2)]$

4.2.2 Turn based timed games

Timed automata can also be considered as arena for turn based timed games. These have been investigated in [30], [32] and also extensively over the weighted counterpart of timed automata. We shall look into [30] and [32] to gain a quick peek into turn based games. These games are very similar in nature to the concurrent ones seen earlier except that the play progresses in a turn based fashion. There is an additional feature in these games. They introduce simple yet relevant metrics into a game and the players now try to minimize or maximize the chosen metric. Min and Max are the two players (counterparts of players 1 and 0 respectively) and while Max is interested in maximizing the value she wins while Min wants to minimize the value she loses. [30] treats this value as the runtime of the play π represented by $rt(\pi)$.

The explicit mention of time divergence does not arise in either of the papers but if we observe the objectives of the players, Max will try to ensure that the runs are time divergent as she wants to maximize the time spent during the play. However, Min might try to make a play zeno as the value she loses is the value gained by Max. But, this state of affairs is no worse than the time divergence condition seen earlier which allows player 1 to cause time convergent plays as long as 0 need not take the blame for it. As it is implicitly achieved, we shall remain silent about this requirement while discussing these papers.

The set of locations L of the timed automaton is partitioned between the two players as it is a turn based game. As it is a timed game moves proposed are (t, a) where t

is the time to be elapsed before the action a. The strategy for a player guides her by determining for a given finite prefix of the play, the move to be proposed.

Reachability time as a value [30]

One of the most interesting yet heavily overlooked metric for a timed game would be the time taken by the play itself. [30] deals with the players working towards this metric. Given a play π, Max wins $rt(\pi)$ while Min loses $rt(\pi)$. Hence it is a zero-sum game with respect to the metric. The winning objective for Max is to ensure that the play reaches a desired set of locations and at the same time maximize the runtime while that for Min would be to avoid the set and minimize the time taken.

As it is important to maximize the time, the authors compute the optimal (best for both players concerned) time to the reachability set $F \subseteq L$ and the number of hops (number of moves) to reach it. These computations are done by two functions T and D which are optimal iff they satisfy all of the following conditions. For a given state $s = (l, v)$

- if $D(s) = \infty$ then $T(s) = \infty$
- if $l \in F$ then $D(s) = T(s) = 0$
- if $l \in L_{min} \setminus F$ then $T(s) = inf_{a,t}\{t + T(S') : s \xrightarrow{t} s'' \xrightarrow{a} s'\}$ and $D(s)$ is $1 + D(s')$ where s' was picked for $T(s)$.
- if $l \in L_{max} \setminus F$ then $T(s) = sup_{a,t}\{t + T(S') : s \xrightarrow{t} s'' \xrightarrow{a} s'\}$ and $D(s)$ is $1 + D(s')$ where s' was picked for $T(s)$.

We have noted in earlier sections that the number of states in a timed automaton are infinite due to the real value clock valuations. This being the case, it would be impractical to calculate the values of T and D for the states individually. As the objective is a reachability one and hence a location goal, we can play the same old trick of working with the untimed regions and then extracting results for the timed version.

Let $\mathcal{R}(\mathcal{A})$ be the region automaton for the original timed automaton $\mathcal{A}$ and let $S_{\mathcal{R}} = L \times \mathcal{R}$ be the set of states in $\mathcal{R}(\mathcal{A})$. Recall that $\mathcal{R}$ is the finite set of regions over the clock valuations. Now we shall define similar functions T_R and D_R for this automaton too. These are known to be optimal iff for a state $s_r \in S_R$

- if $s_r \in F_R$ then $D_R(s_r) = T_R(s_r) = 0$
- if $s_r \in S_{min} \setminus F_R$ then $T_R(s_r)$ is the minimum value of $T(s)$ such that $s_r = [s]$ taken over all possible ways out to reach the next region and $D_R(s_r)$ is incremented by 1 in the direction chosen by T_R.
- if $s_r \in S_{max} \setminus F_R$ then $T_R(s_r)$ maximizes the value of $T(s)$ such that $s_r = [s]$ taken over all possible ways out to reach the next region and $D_R(s_r)$ is incremented by 1 in the direction chosen by T_R.

Now we will see a strategy improvement algorithm for computing these functions and the winning strategy for Max.

1. Start with any regionally constant positional strategy for player μ_o for Min
2. Let $i = 0$
3. Solve the subgame restricted to μ_i as a one player game to obtain T_i, D_i. *As Min does not have any moves, the game is played solely by player Max.*
4. Improve μ_i to be the most optimal for player Min at all states if not already optimal.
5. $i = i + 1$
6. stop when $\mu_{i+1} = \mu_i$
7. return T_i, D_i

The algorithm to solve the one player game for Max is along the same lines and can be obtained by replacing all occurrences of Min above by Max. Note that though positional strategies do exist as the goal is reachability, they might require a finite amount of memory to keep track of time delay to be proposed (which is typically obtained from the clock intervals serving as delay between two regions).

Complexity - The algorithm takes $O(|S_{\mathcal{R}}|)$ number of iterations to terminate and hence runs in $O(|S_{\mathcal{R}}|^3)$ time. The problem of solving reachability time games in EXPTIME-complete on timed automata with at least two clocks.

Similar work has been done independently for concurrent timed games too [31].

Average time as the value [32]

In this game the metric the players would be fighting for is the average time per transition and this in itself is the sole winning objective. Untimed versions of these games were studied [41], [42] and [43]. The metric in an untimed game is the price attached to the edges while the players play in a turn based fashion. While Min tries to minimize the average (taken over the infinite play) price of the play, Max tries to maximize it. For a price function $price$ and a play $\pi = v_0 v_1 \cdots$, $V_*(\pi) = lim\ sup_{n\to\infty}.(1/n).\sum_{i=1}^{n} price(v_{i-1}, v_i)$ is the value lost by Min. The value V^* gained by Max is obtained by replacing the sup by inf. For a vertex v, $\underline{v} = inf_{\mu\in\Sigma_{min}}\ sup_{\chi\in\Sigma_{max}} V^*(outcome(v,\chi,\mu))$ while $\overline{v} = sup_{\chi\in\Sigma_{max}}\ inf_{\mu\in\Sigma_{min}}\ V_*(outcome(v,\chi,\mu))$. A game is said to be determined if for all v, $\overline{v} = \underline{v}$. [43] shows that every average-price game is determined and optimal positional strategies exist for both players.

In the timed setting, the metric is the average runtime with the average taken over several plays with the same strategies. For player Min, $V_*(s, \chi.\mu) = lim\ sup_{n\to\infty}.(1/n).$ $rt(outcome(s, \chi.\mu))$ is the value lost for plays starting at state $s = (l, \nu)$ with Max and Min using the strategies χ and μ respectively. Similarly $V^*(s, \chi, \mu)$ can be defined by replacing sup with inf. As before, $\overline{s}$ and $\underline{s}$ can be defined. The notion of determinacy remains same when interpreted over the infinite state set of states (l, ν) where l is the

location of the timed automaton and ν is the clock valuation.

Due to the infinitely many states, we rely once again on the strength of region automaton. The definitions of all the values V_*, V^*, $\overline{s_r}$ and $\underline{s_r}$ are similarly defined except that the runtime rt would calculate the time elapsed between two regions as opposed to two states. Now, positional determinacy in the game on region automata follows from the result in the untimed average-price games and the fact that the region set is finite. The average-time game on the region automaton is solved using the known techniques for untimed versions. Thus the game on timed automaton is determined and needs some memory to maintain timing information.

Complexity - The authors claim that the value of the game on region automaton can be solved by an EXPTIME algorithm. Additionally, it is EXPTIME-complete for timed automata with at least two clocks.

Chapter 5

Problems to ponder over

We have studied several different algorithms for solving untimed parity games in Chapter 3 and several formulations of timed games in Chapter 4. While the untimed game scenario is quite mature and stabilised with most of the resarch there focusing mainly on improving the complexity. Timed games are yet to reach that stage of maturity. There are several directions of research yet to be given attention to. We shall list a few problems that we have formulated over the period of this literature study. We are as yet not aware of any existing solutions for these problems and equally unware about the feasiblity and the impact factor.

Parity games

Following is a list of open problems yet to be explored for parity games.

1. **Weak Parity Games [18]** - Does the complexity remain polynomial for a bounded number of visits to the max or min priority vertex? Of course, the strategies will no longer be memoryless. How does the complexity change with the number of visits?

2. **Complexity reducing tricks for parity games** - Parity games are known to enjoy memoryless determinacy [4] and hence all the strategies every built are memoryless. The best known algorithm is sub-exponential [6]. Will the complexity reduce if we considered strategies with finite or infinite memory? So far, very few algorithms use the trick of randomization. Can there be a randomized algorithm which achieves better complexity?

3. **Restricted parity games** - with some of the parameters (number of priorities, out edges, in edges, no. of vertices, no. of visits, etc) restricted can we reduce the complexity.

4. **Multi player parity games** - definition of the game has to be proposed. Questions similar to parity can be asked here too.

5. **Infinite arena games** - can the currently known techniques (progress measures [12], strategy improvement [14]) be extended to infinite arena games?

6. **Nash equilibrium, Nash determinacy** - untimed games with the payoff being simple metrics like number of times minimum vertex visited, number of vertices in the path before the loop/desired vertex, number vertices in the loop i.e; we want to see the desired vertex as often as possible etc.

7. **Sharing of vertices i.e;** $V_0 \cap V_1 \neq \emptyset$ - pick some criterion to decide whose move is picked. This should be interesting cause it is too strict to declare that only one player has a say. At some vertices it is turn based and at others it is concurrent i;e an open system which is asynchronous most of the times but has some synchronous points?

8. **Deterministic sub-exponential algorithm for solving parity games [6]** - based on dominions of size $l = \lceil \sqrt{2n} \rceil$. How does tweaking this factor affect the complexity given by recurrence relation $T(n) \leq O(n^l) + T(n-1) + T(n-l)$?

9. **Probabilistic turn based parity games** - each player has a probability distribution over the outgoing edges from her vertex.

Timed games

Following are some directions which can be explored in the context timed games.

1. **Concurrent probabilistic times games** - enriching games in [24] with time.

2. **Complexity of timed buchi games** - traditional timed automata with locations partitioned among the players i.e; turn based games. The complexity should be lower than that for parity conditions technically speaking. The same technique of [29] can be used to untime and then apply untimed Buchi games algorithms. In untimed case, they exhibit better complexity than parity conditions.

3. **Problems of parity with time** - add timing dimension to parity problems in earlier section.

4. **Coalitional timed games** - non-zero sum games.

5. **Hybrid arena for timed games** - considering some of the locations as concurrent and some as turn based. Same as problem 7 in untimed parity games problems.

6. **Arena defined on the states as opposed to locations** - For example one of the ways to define the arena on states would be to mention for a location, the invariants under which player i can choose the outgoing edge.

7. **Timing constraints in the goals** - so far most of the research has been aimed at location goals.

8. Do the results vary with the number of clocks - as the results are closely tied to region equivalence, would the number of clocks change the known results.

9. **Equilibria** - There are some papers dealing with time as a metric, can we formulate notions of equilibria for these games?

Bibliography

[1] R. Alur and D.L. Dill, A Theory of Timed Automata, *Theoretical Computer Science*, 126(2), 1994.

[2] E. Grädel, W. Thomas and T. Wilke, Automata, Logics and Infinite games - A guide to current research, Springer, 2002.

[3] M. Huth and M. Ryan, Logic in computer science, Cambridge University Press, 2004.

[4] R. McNaughton, Infinite games played on finite graphs, Annals of Pure and Applied Logic, 65 (1993), pp. 149 - 184.

[5] T.A.Henzinger and V.S.Prabhu, Alternating time temporal logic, FORMATS 2006: pp. 1-17.

[6] M.Jurdziński, M. Paterson and U. Zwick, A deterministic subexponential algorithm for solving parity games. SODA 2006: pp. 117-123, 2006

[7] W. Ludwig, A subexponential randomized algorithm for the simple stochastic game problem, information and Computation, 117 (1995), pp. 151 - 155.

[8] H. Björklund, S. Sandberg, and S. Vorobyov, A discrete subexponential algorithm for parity games, in Symposium on Theoretical Aspects of Computer Science, STACS 2003, vol. 2607 of LNCS, Springer, 2003, pp. 663 - 674.

[9] J. Matoušek, M. Sharir, and E. Welzl, A subexponential bound for linear programming, Algorithmica, 16 (1996), pp. 498 - 516.

[10] G. Kalai, A subexponential randomized simplex algorithm (Extended abstract), in Symposium on Theory of Computing, STOC92, ACM Press, 1992, pp. 475 - 482.

[11] W. Zielonka, Infinite games on finitely coloured graphs with applications to automata on infinite trees, Theoretical Computer Science, 200 (1998), pp. 135 - 183.

[12] M.Jurdziński, Small progress measures for solving parity games, STACS 2000: pp. 290 - 301, 2000.

[13] Luca de Alfaro, Marco Faella: An Accelerated Algorithm for 3-Color Parity Games with an Application to Timed Games. CAV 2007: 108-120.

[14] J. Vöge and M.Jurdziński, A discrete strategy improvement algorithm for solving parity games. CAV 2000: pp. 202 - 215, 2000.

[15] A. Hoffman and R. Karp, On nonterminating stochastic games, Management science, 12:359-370, 1966.

[16] A. Condon, On algorithms for simple stochastic games, In Jin-Yi Cai, editor, Advances in Computational Complexity Theory, vol 13 of DIMACS Series in Discrete Mathematics and Theoritical Computer Science, pages 51-73, American Mathematical Society, 1993.

[17] Anuj Puri, Theory of Hybrid Systems and Discrete Event Systems, PhD thesis, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, December 1995. Memorandum No. UCB/ERL M95/113.

[18] K. Chatterjee, Linear time algorithm for weak parity games. CoRR abs/0805.1391: (2008).

[19] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, Handbook of Formal Languages, volume 3, Beyond Words, chapter 7, pages 389 - 455. Springer, 1997.

[20] K. Chatterjee, T.A. Henzinger and N. Piterman, Generalized parity games. FoSSaCS 2007: 153-167, 2007.

[21] N. Piterman and A. Pnueli. Faster solution of rabin and streett games. In LICS, pages 275 - 284, IEEE, 2006.

[22] E.A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In FOCS, pages 328 - 337, IEEE, 1988.

[23] E.A. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In FOCS, pages 368 - 377, IEEE, 1991.

[24] Luca de Alfaro, Thomas A. Henzinger: Concurrent Omega-Regular Games. LICS 2000: 141-154.

[25] B. Bérard and C. Duford, Timed automata and additive clock constraints, *Information Processing Letters*, 75(1-2): 1-7 2000.

[26] P. Bouyer, C. Duford, E. Fleury, and A. Petit, Updatable Timed Automata, *Theoretical Computer Science*, 321(2-3): 291-345, 2004.

[27] L. Manasa, K. S. Narayanan and K. Nagaraj, Updatable Timed Automata with Additive and Diagonal Constraints, *Proceedings of CIE'08, LNCS*, p 410-417, 2008.

[28] T. Brihaye, V. Bruyère and J. Raskin, Model-Checking for Weighted Timed Automata, *Proceedings of FORMATS/FTRTFT 2004*, 277-292, 2004.

[29] K. Chatterjee, T.A. Henzinger and V. S. Prabhu, Timed parity games : Complexity and Robustness, FORMATS 2008, LNCS 5215, pp 124-140, 2008.

[30] A. Trivedi, Reachability-time Games, CoRR abs/0907.3414: (2009).

[31] T. Brihaye,T. A. Henzinger, V. S. Prabhu and J. Raskin, Minimum-Time Reachability in Timed games

[32] M.Jurdziński and A. Trivedi : Average-Time Games, FSTTCS 2008

[33] L. de Alfaro, M. Faella, T. A. Henzinger, R. Majumdar, M. Stoelinga, The Element of Surprise in Timed Games. CONCUR 2003: 142-156

[34] K. Chatterjee, T. A. Henzinger, Vinayak S. Prabhu: Trading Infinite Memory for Uniform Randomness in Timed Games. HSCC 2008: 87-100

[35] O. Maler, A. Pnueli and J. Sifakis : On the synthesis of discrete controllers for timed systems (an extended abstract). STACS 1995. LNCS, vol. 900, pp. 229 - 242. Springer, Heidelberg (1995)

[36] M. Faella, S. La Torre, A. Murano, Dense Real-Time Games. LICS 2002: 167-176

[37] A. Pnueli, E. Asarin, O. Maler and J. Sifakis, Controller synthesis for timed automata.In: Proc. System Structure and Control, Elsevier, Amsterdam (1998)

[38] E. Asarin, O. Maler , As soon as possible: Time optimal control for timed automata. HSCC 1999. LNCS, vol. 1569, pp. 19 - 30. Springer, Heidelberg (1999)

[39] T. A. Henzinger, P.W. Kopke, Discrete-time control for rectangular hybrid automata. Theoretical Computer Science 221, 369 - 392 (1999)

[40] L. de Alfaro, T. A. Henzinger and R. Majumdar, Symbolic Algorithms for Infinite-State Games. CONCUR 2001: 536-550

[41] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. Theoretical Computer Science, 158:343 - 359, 1996.

[42] J. Filar and K. Vrieze. Competitive Markov Decision Processes. Springer, 1997.

[43] T. Liggett and S. Lipman. Stochastic games with perfect information and time average payoff. SIAM Review, 11:604 - 607, 1969.

[44] P.J.G. Ramadge and W.M. Wonham. The control of discrete-event systems. IEEE Transactions on Control Theory, 77:81 - 98, 1989.

[45] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In Proc. Symp. Principles of Programming Languages, pages 179 - 190. ACM, 1989.

[46] K. Chatterjee, T. A. Henzinger, and M. Jurdziński, Games with Secure Equilibria, FMCO 2004, LNCS 3657, pp. 141 - 161, 2005.

[47] M. Ummels. Rational Behaviour and Strategy Construction in Infinite Multiplayer Games. Diploma thesis, RWTH Aachen, 2005.