

Compositional Synthesis using Admissible Strategies

*A B.Tech Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of*

Bachelor of Technology

by

Aman Raj
(170101006)

Mayank Wadhwani
(170101038)

under the guidance of

Dr. Purandar Bhaduri



to the

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM**

CERTIFICATE

*This is to certify that the work contained in this thesis entitled “**Compositional Synthesis using Admissible Strategies**” is a bonafide work of (**Roll No. 170101006 and 170101038**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Dr. Purandar Bhaduri**

Professor

Department of Computer Science & Engineering

Indian Institute of Technology Guwahati, Assam

Acknowledgements

We would like to express our gratitude to our supervisor **Dr. Purandar Bhaduri** for giving us the opportunity to explore this new field of Infinite Games and the much needed zest to delve into some of the state-of-the-art works.

Contents

1 Abstract & Introduction	1
1.1 Abstract	1
1.2 Introduction	2
1.3 Organization of The Report	4
2 Preliminaries	5
2.1 Arena	5
2.2 Controlled Predecessor	6
2.3 Attractor	6
2.4 Multiplayer Games	7
2.5 Reachability Games	7
2.6 Safety Games	8
2.7 Strategy	8
2.8 Winning Strategy	8
2.9 Dominance for strategies	9
2.10 Value	9
3 Methodology	11
3.1 Algorithms Used	11
3.1.1 Checking for Winning Strategy	11
3.1.2 Computing Admissible Strategies	13

3.2 Our Idea	14
4 Results	16
4.1 Example 1	16
4.2 Example 2	17
5 Applications	19
5.1 Scheduler	19
5.2 Burger King Robot Delivery System	21
6 Conclusion and Future Work	25
6.1 Results	25
6.2 Future Work	25
References	27

Chapter 1

Abstract & Introduction

1.1 Abstract

Controller Synthesis revolves around the design and implementation of systems working in an environment to satisfy their goals. A popular approach to solve controller synthesis is to find winning strategies in graph games for the system against the environment. **Safety games** are a class of graph games in which the winning objective for a player is to never visit their corresponding unsafe states. However, in several cases where no winning strategies exist, we find **admissible strategies** in which players play rationally instead of being adversarial and achieve their goals. Compositional Synthesis refers to the process of realizing a controller for the entire system by breaking it into several controllers which would represent individual components. In our work, we make an attempt to design component-based controllers using Admissible strategies in multiplayer games.

1.2 Introduction

Games played on graphs with finite vertices have been a topic of study for various years with applications lying in a plethora of examples like controller synthesis. Given, a model of the assumed behaviour of the environment and a system goal, controller synthesis aims at producing a behavioural model for a component that when executing in an environment consistent with the assumptions results in a system that is guaranteed to satisfy the goal. An approach to solve this controller synthesis problem involves graph games.

The **controller synthesis** problem for reactive systems can be modelled into an interactive game of strategy typically between two players, *Player 0* being the system and *Player 1* being its environment. These so-called **Graph Games** are played infinitely on a finite graph since there are no dead ends. Vertices are partitioned between players and the player owning the vertex decide the next move from that vertex.

There are different winning conditions depending on the required specification of the controller. It could be reachability, safety, etc. A strategy followed by a player is the mapping of moves the player makes on each of its vertex. If a strategy allows the *Player 0 (the system)* to satisfy the required specification or achieve his goals, no matter what strategy the opponent *Player 1 (the environment)* follows, then it is called a *winning strategy* for *Player 0*.

Safety Games are a class of graph games which involves safety objectives. The condition in safety games is that *Player 0* is not allowed to enter a set of unsafe/bad vertices, i.e., the whole game play should be confined to a set of safe vertices. To find a winning strategy for *Player 0* in safety games, we make use of its duality with Reachability games, convert the safety objective into reachability objective and implement the algorithm used to find the winning strategy for reachability games.

But, there exists many scenarios of safety games where no winning strategy exists for *Player 0*, in such cases, we look for best possible strategies which are not dominated by the opponent. These strategies are called *admissible strategies* or *non-dominated strategies*.

Generally, to find a winning strategy, both players play in an adversarial manner, but admissible strategies include those ones in which *Player 1* plays **rationally** or co-operates and focuses on achieving its goal and if possible, without dominating *Player 0*. This way, we can find strategies which could lead to a win-win situation for both the players. In other words, *Player 1* helps *Player 0* win as long as he is winning.

Compositionality refers to the property that the results observed for the individual components can be used for the entire system. So, if we break the system into several components and find appropriate results for each of them, we would then be able to find the results for the system.

Compositional Synthesis helps in realizing the controller for a given system by breaking it into multiple components and obtaining results for each of them. Much work has been done on compositional synthesis by finding the winning/dominant strategies for the players. This is because the winning strategies are compositional i.e. if each component can guarantee to satisfy its objective irrespective of how the other players move, then the entire system would be able to achieve all its objectives. However, this may not always be the case as there may exist scenarios where winning strategies may not exist. In our work, we try to realize component-based controllers using admissible strategies in multiplayer games, i.e. for each player, we would try to find admissible strategies that would be winning against all admissible strategies for all other players. We would also assume that each player may have their own independent objective unlike the case where all players have the same objective as proposed in [DF14].

There are several advantages that we would get because of using admissible strategies. The set of Assume Admissible (AA) profile is **rectangular**, i.e. : any combination of AA-winning strategies independently chosen for each player is an AA-winning profile. Also, it follows the **robustness** property. If we replace some subset of strategies in an AA-winning profile by other set of arbitrarily admissible strategies, the other players would still be able to satisfy their objectives.

Moreover, we would be using the process of **iterated elimination of dominant strategies** which helps us obtain *admissible strategies* for the safety games with no winning strategy in the paper.

1.3 Organization of The Report

This chapter introduces the problem statement covered in this report. We provided a brief description of the topic and talked about the importance of controller synthesis in real life. The rest of the chapters are organised as follows: next chapter we discuss all the preliminaries that would help brush up the keywords required. In Chapter 3, we describe the methodology that we are following, we would briefly discuss the algorithms used and the idea that we worked on. In Chapter 4 and 5, we present results as found in various examples and look at some real life applications where our work can be used. And finally in chapter 6, we conclude with some future works.

Chapter 2

Preliminaries

The following section contains a background of the different terminologies that will be used in the later sections. We will be referring to the game described in Fig 4.1.

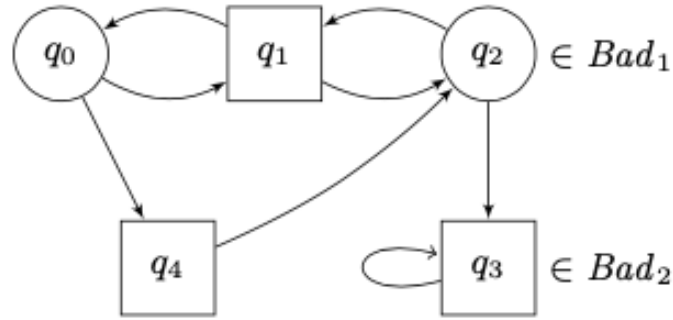


Fig. 2.1: Example of a Game

2.1 Arena

An arena is defined as a tuple:

$$A = (V, V_0, V_1, E).$$

Here, V represents the total set of all vertices in the graph. V_0 represents the set of vertices for the first player (denoted by Player 0), V_1 denotes the set of vertices for the second player and finally E denotes the set of edges between 2 vertices of the graph.

Informally, an arena is a directed graph which contains edges between vertices representing the different players playing the game. In the example, the q_0 and q_2 will belong to the set V_0 and the remaining would belong to the set V_1 .

2.2 Controlled Predecessor

Given a player i and some set R , the controlled predecessor $CPre_i(R)$ of R is defined as:

$$CPre_i(R) = \{v \in V_{1-i} \mid \forall \alpha \in R, (v, \alpha) \in E\} \cup \{v \in V_i \mid \exists \alpha \in R, (v, \alpha) \in E\}$$

Informally, the controlled predecessor returns the set of all vertices from which if we start, we are bound to reach some vertex of the given set R .

So we break this problem into two, finding all vertices that belong to same set of the player and check if there is any vertex from which we can reach R , if yes, we simply add the vertex to the set.

Now, if the vertex doesn't belong to the player's vertices, then we check if all the outgoing edges from this vertex would eventually lead to R . If it does, the second player will have no option but to visit the R .

2.3 Attractor

Given, a *Player* i and some set R for a reachability game, the *i-attractor* given by $Attr_i(R)$ of set R in arena A is defined inductively using the controller predecessors defined in section 2.2 as follows:

- $Attr_i^0(R) = R$
- $Attr_i^{n+1}(R) = Attr_i^n(R) \cup CPre_i(Attr_i^n(R))$
- $Attr_i(R) = \bigcup_{n \in \mathbb{N}} Attr_i^n(R)$

While running this process, after some iterations, no new vertex gets added in the Attractor. Generally, after at most $|V|$ steps, the stages of attractor converge and become stationary.

Winning region for *Player i* refers to the set of vertices of the arena from where *Player i* has a winning strategy. Hence, to get $Attr_i(R)$ which gives the winning region for the *Player i* denoted by $W_i(G)$, we need to compute only $Attr_i^{|V|}(R)$.

2.4 Multiplayer Games

A multiplayer game G is defined as a tuple:

$$G := \langle P, A, (Win_i)_{i \in P} \rangle$$

Here, P refers to the non-empty set of players involved in the game. A represents the arena of the game as defined in section 2.1. Win_i denotes the winning condition for each player i in the set players P .

In this paper, we will be focussing on 2-player games. So, $P = \{0, 1\}$. We will be looking at the games from perspective of *Player 0 (system)*. Depending on the objective of the game, the winning condition could be a reachability condition, safety condition, etc.

2.5 Reachability Games

A class of multiplayer games with reachability as its winning condition. The reachability condition $REACH(R)$ is defined as follows:

$$REACH(R) := \{\rho \in V^\omega \mid Occ(\rho) \cap R \neq \emptyset\}$$

Here, R called the reachability set is a set of vertices such that $R \subseteq V$. V refers to the total vertices of the arena. V^ω denotes various plays or sequences of moves possible in the game. $Occ(\rho)$ denotes the vertices reached atleast once in the play.

The aim of *Player 0* is to reach at least one vertex from a set of vertices R once. Hence, the set of vertices common between $Occ(\rho)$ and R should not be null/empty. *Player 1*

shows adversarial play and tries to avoid him from doing so.

2.6 Safety Games

Safety objective is another type of winning objective involved in multiplayer games. This class of games have a safety condition $SAFETY(S)$ defined as follows:

$$SAFETY(S) := \{\rho \in V^\omega \mid Occ(\rho) \subseteq S\}$$

Here, S is a set of safe vertices such that $S \subseteq V$. V refers to the total vertices of the arena. V^ω denotes various plays or sequences of moves possible in the game. $Occ(\rho)$ denotes the vertices reached at least once in the play.

The aim of *Player 0* is to remain confined in S always. In other words, for *Player 0* to achieve its objective, at no point in the play, a non-safe or bad vertex from the set $V \setminus S$ should be reached. Hence, the set of vertices in $Occ(\rho)$ should be a subset of safe vertices S . *Player 1* shows adversarial play and tries to avoid him from doing so.

If we observe this game from perspective of *Player 1*, it can be seen as a *reachability game* where the objective of *Player 1* is to reach any vertex from the set $V \setminus S$. This gives us the sense of *duality* between these two types of games.

2.7 Strategy

A strategy for a *Player i* in a game given by the arena $A = (V, V_0, V_1, E)$ is a function $\sigma : V^*V_i \rightarrow V$, such that $\sigma(wv) = v'$ where $w \in V^*$ and $v \in V_i$ and $(v, v') \in E$. Hence, σ represents the mapping of states to moves chosen by the *Player i* against *Player 1-i*.

2.8 Winning Strategy

Given, a game $G = (P, A, (Win_i)_{i \in P})$, a strategy σ is called a winning strategy for the *Player i* from a vertex $v \in V$ if every play starting from vertex v following the strategy

σ satisfies the winning condition Win_i for that player, irrespective of what strategy is followed by the opponent.

For safety games, a winning strategy σ for *Player 0* from vertex v makes sure that if a play starts from vertex v following that strategy, no unsafe vertex is reached during the play irrespective of what moves are chosen by *Player 1*.

2.9 Dominance for strategies

Given, a rectangular set of strategy profiles $S = \prod_{i \in P} S_i$ where S_i represents a set of strategies of *Player i*.

A strategy σ is said to very weakly dominate another strategy σ' w.r.t. S , expressed as $\sigma \succsim_S \sigma'$, if from all possible states s , the following condition satisfies:

$$\forall \tau \in S_{-i}, Win_i^s(\sigma', \tau) \implies Win_i^s(\sigma, \tau)$$

A strategy σ is said to weakly dominate another strategy σ' w.r.t. S , expressed as $\sigma \succ_S \sigma'$, if the following conditions satisfies:

- $\sigma \succsim_S \sigma'$
- $\neg(\sigma' \succ_S \sigma)$

Here, the strategy σ' is said to be dominated in S as σ dominates it. And a strategy which is not dominated by any other strategy is an *admissible strategy* in S .

To obtain the set of admissible strategies, we iteratively eliminate the dominated strategies.

2.10 Value

After n th step of elimination of dominated strategies, the value of history h for *Player i* is defined as:

- if \exists a winning strategy from $\text{last}(h)$, then $Val_i^n(h) = 1$.
- if \nexists winning strategy from $\text{last}(h)$ even if the other player helps, then $Val_i^n(h) = -1$.
- in all other cases $Val_i^n(h) = 0$.

Informally, the value of a state for a player denoted by $val_i^n(s)$ is 1 if there is a winning strategy for the player from that state. This means that even if the second player plays in an adversarial fashion, the player **will still** end up winning the game.

Moreover, the value of 0 means that the player **will always lose** from this state **even if** the second player plays in favour of the player. For all the remaining cases, we assign the value 0.

Chapter 3

Methodology

So far, we have looked at the basic terminologies that would be used in the rest of the paper. We now turn our attention towards our main idea. We start by briefly discussing the algorithms used in the implementation. This includes algorithms that were used to find winning strategies in safety games and also the algorithm used to find admissible strategies. After that, we briefly comment on the idea that we have worked on.

3.1 Algorithms Used

In this section, we briefly discuss about the algorithms that were employed in our implementation.

3.1.1 Checking for Winning Strategy

We discuss the algorithm [1](#) in this section which will be used to check if a winning strategy exists from a given initial state for a safety game.

We first convert our safety game into its corresponding **dual**, the reachability game, in the process, all vertices belonging to player 0 in the safety game will now belong to player 1 in the reachability game and similarly for the other player also. As discussed above, player 0 will win the safety game if its dual ie. player 1 wins the reachability game or in other

Algorithm 1: Check if winning strategy

Input: Arena $A = (V, V_0, V_1, E)$, Initial State, Bad States
Output: True if winning strategy exists from given state
Find Dual of given Input Graph (Assign all vertices for player 1 to player 0 and vice versa);
attractor_i = Good_States (Initially the given good states act as the Attractor);
while *Length of attractor_i changed from previous iteration* **do**
 Find $CPre_i(\text{attractor_i})$;
 Updated_Attractor_i = attractor_i \cap $CPre_i(\text{attractor_i})$;
 attractor_i = Updated_Attractor_i;
end
if *initial_state not in attractor_i* **then**
 return True;
else
 return False;
end

Algorithm 2: Compute Iteratively Admissible Strategies

Input: Arena $A = (V, V_0, V_1, E)$, The winning conditions of each player (in this case set of bad states for each player) Win_i
Output: Set of admissible strategies for the players
while $\exists i \in P, T_i^n \neq T_i^{n-1}$ **do**
 for $s \in V$ **do**
 if \exists winning strategy for player i from s in graph **then**
 $Val_i^n = 1$;
 else if \nexists winning strategy for player i from s even if the other player helps in graph **then**
 $Val_i^n = -1$;
 else
 $Val_i^n = 0$;
 for $i \in P$ **do**
 $T_i^n = T_i^{n-1} \cup \{(x, y) \in E \mid x \in V_i \wedge Val_i^n(x) > Val_i^n(y)\}$
 $n = n + 1$;
 graph = graph $\setminus T^{n-1}$

words player 0 loses the reachability game. It should be noted that player 0 of the safety game is **not** the same as the player 0 of the reachability game.

Now, to find if the player 0 will win the safety game, player 0 should lose the reachability game or the initial state should not be a part of the final value of `attractor_i`. This is because the final value of `attractor_i` denotes the winning region for player 0, ie. the set of all vertices from which we will eventually reach some vertex \in good states and if the initial state is not a part of this winning region, it means player 0 will lose the reachability game which is desired.

We now focus our attention towards finding the final value of `attractor_i` or the winning region for a reachability game. We can do this by looping till there is no more changes in the `attractor_i` and at each iteration finding the controlled predecessor as described above. We take the union of the found controlled predecessor with our `attractor_i` and if there is no change break the loop.

In this manner, we are able to find if there exists a winning strategy for a player.

NOTE: The algorithm described in [RBS14] also requires us to find if from a state we will always lose, that is even if the other players help us, we are still bound to lose.

We have in our implementation solved this by a simple idea. We convert **all vertices** to player 0, ie. we assume that there is no 'other player' and so if there is only one player, it will not play adversarial to itself. A player will never want to defeat itself. So if a winning strategy exists for the player from a given state, this means that there is still chances of the player to win the game but if no winning strategy exists for this case, we declare that the player will lose the game no matter how optimally it plays.

3.1.2 Computing Admissible Strategies

In this section, we briefly discuss about the second half of our algorithm where we find the admissible strategies for a given safety game. We do this by taking references from [RBS14]. We initialize the T_i set to ϕ and loop till we find ourselves in a condition where for all states

of the graphs, the value of the set T did not change for an iteration, that is the set T has converged.

In a particular iteration, we iterate over all the vertices of the graph and compute the values of the state for the player i . We make use of the algorithms as discussed in the previous section. So we can compute whether or not a winning strategy exists or it never exists which corresponds to the values 1 and -1 respectively. If both of these are false, we simply assign the value 0 and continue.

We then for all players, iteratively construct the **T set**, by adding all those edges to this set where the value of start vertex is greater than the value of successor, ie. all those edges where we will move from a higher value to a lower value. We then remove all the edges present in this T set from the graph as we will never take those paths.

We continue this process till the above mentioned condition is reached and then we break the loop. In the end, we have the set of all admissible strategies for the players.

3.2 Our Idea

We have presented algorithms that helps in determining winning strategies and admissible strategies in a given graph game. We wish to expand this idea and try to make use of these algorithms in real life scenarios. We wish to apply them to design controllers which would have cooperative components with an adversarial environment. The individual objectives of each component are satisfied by following admissible strategies (strategies that would allow the environments to be cooperative). In other means, we would try to find a strategy profile $\sigma_p = (\sigma_1, \dots, \sigma_n)$ such that $G, \sigma_p, \sigma_e \models \bigwedge_{i \in P} \phi_i$ for all strategies σ_e of the environment.

In other words, this means that we have to compute a strategy profile such that each individual strategy is admissible and is winning against other admissible strategies of the other components in the profile and any strategy of the environment.

So we would be using admissible strategies to find if the objectives for all the players can be satisfied even if the environment plays in an adversarial fashion.

This may lead to a scenario where there are multiple possible strategies possible from a node. In such a case, we would try to think of a best effort strategy, i.e. choose such a strategy that would help us give best results. We would discuss briefly on this in Chapter 5 where we have provided real life applications of this idea.

Chapter 4

Results

In this section, we present the results that was observed when we run our algorithm on different types of graphs as inputs. These graphs were run to test the functioning of our algorithms.

4.1 Example 1

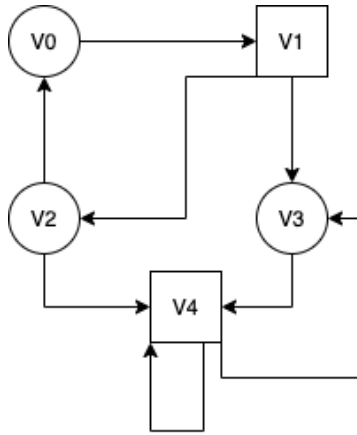


Fig. 4.1: Example 1

In this example we have $v3 \in Bad_0$ and $v4 \in Bad_1$ or visiting v_3 will defeat player 0 and visiting v_4 will defeat player 1.

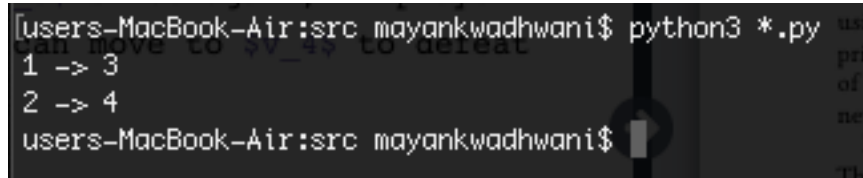
It can be seen that there **does not** exists any winning strategies for the players. So we

find the admissible strategies using our algorithm. Our implementation is such that we print the set of edges that is to be deleted from the total set of edges. In other words, we print the edges that we would never visit or the final T set as defined above.

It was expected that we should remove the edge $v_2 \rightarrow v_4$. This is because if the player 0 plays this move and reaches v_4 , player 1 will lose the game, and player 1 from this point can act adversarial and move to v_3 which will defeat player 0 also. So player 0 instead will prefer moving to the vertex v_0 .

We also expect to remove the edge $v_1 \rightarrow v_3$ since again, if player 1 moves to v_3 and defeats v_3 , player 0 can move to v_4 to defeat player 1.

So we expected to remove the edges $v_1 \rightarrow v_3$ and $v_2 \rightarrow v_4$ and our actual results were **in accordance** with the expectations as can be seen from Fig 4.2.



```

[users-MacBook-Air:src mayankwadhvani$ python3 *.py
1 -> 3
2 -> 4
users-MacBook-Air:src mayankwadhvani$

```

Fig. 4.2: Result for Example 1

4.2 Example 2

In this example we have $v_2 \in Bad_0$ and $v_3 \in Bad_1$ i.e., visiting v_2 will defeat player 0 and visiting v_3 will defeat player 1.

Again, in this example also, it can be verified that player 0 does not have a winning strategy. We then try to compute the admissible strategies for the same.

It can be noticed that the *value* of q_4 initially for player 1 is -1 since if it reaches q_2 and defeats player 0, player 0 can act adversary and defeat it. But its *value* from v_0 is 0, since if player 0 moves to q_1 , we can move back to q_0 which will form a cycle of length 2. So since value of q_0 is greater than the value of q_4 for player 1, we are expected to remove this edge.

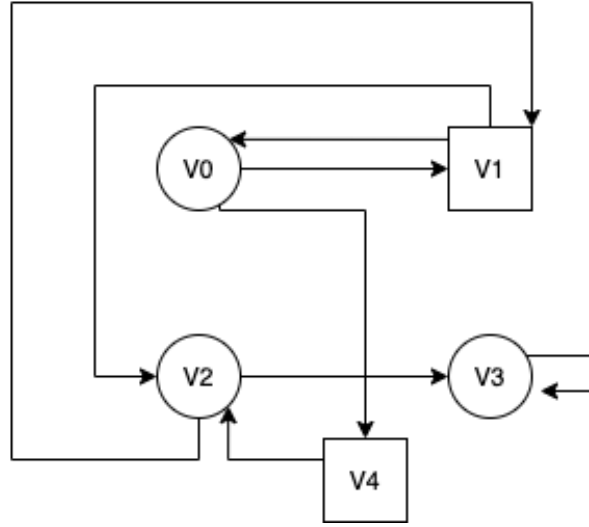


Fig. 4.3: Example 2

In a similar fashion, one can argue to expect $v_1 -> v_2$ getting removed since if player 1 moves to q_2 and defeats player 0, it can move to q_3 and defeat player 1.

So we expected to see two edges $v_0 -> v_4$ and $v_1 -> v_2$ getting removed. This is what was observed after running our implementation.

```

[users-MacBook-Air:src mayankwadhvani$ python3 *.py
0 -> 4
1 -> 2
users-MacBook-Air:src mayankwadhvani$

```

Fig. 4.4: Result 2

We can therefore conclude that **our implementation** of finding admissible strategies for given safety games **yields corrects outputs**.

Chapter 5

Applications

In this chapter, we shall look at a examples which depicts few real life applications of controllers that could be designed by using admissible strategies. We start by the example of a real time scheduler that is represented as a three player game. Next, we discuss about a robot delivery problem that is represented as a two player game. We represent both these examples as an infinite graph game and find admissible strategies to allow all the players to reach their goals. These admissible strategies are found using the methods discussed in Chapter 3. We shall look at the real time scheduler example now.

5.1 Scheduler

This is a real life application of synthesis of a scheduler with two tasks in hand described in [\[RBS16\]](#). There are three players involved in the system: **User**, **Controller** and **Scheduler**. The overview of the system includes: User can send two different actions a_1 and a_2 to Controller. Upon receiving any action a_i , Controller must issue a corresponding request r_i to Scheduler. The task of Scheduler is to schedule event q_i corresponding to each received request r_i . There are some constraints attached with Controller and Scheduler which they have to meet while fulfilling these requests.

On modelling this system into a multiplayer game, each round involves three steps (one

for each player) starting with User, then Controller followed by Scheduler. To send an action a_i , User will set it to *true* in his turn. Similarly, Scheduler and Controller can set their corresponding boolean variables in their turn to send requests and schedule events respectively.

The objective of User is trivial, i.e., accepting for all outcomes. The objective of Controller is to send request r_i within k rounds, after receiving any action a_i . Scheduler has a similar objective, to schedule event q_i within t rounds on getting request r_i .

Expressing the requirements of the system in the form of LTL gives us following:

- $\phi_{User} = true$
- $\phi_{Controller} = G(a_1 \implies F_{\leq k} r_1) \wedge G(a_2 \implies F_{\leq k} r_2)$
- $\phi_{Scheduler} = G(r_1 \implies F_{\leq t} q_1) \wedge G(r_2 \implies F_{\leq t} q_2)$

Talking about the solution of this game, there exists no winning strategy for both Controller and Scheduler. For any strategy of Controller, the Scheduler can constantly play false without scheduling any event and hence r_i will remain true, hence no more requests can be made by Controller on getting actions. For any strategy of Scheduler, if the time bounds are favourable, Controller can keep on sending requests and Scheduler won't be able to satisfy the requirements. Hence, we look for admissible strategies to satisfy the objectives of all the players in the system.

Converting the above mentioned requirements into safety objectives gives us a 3-player safety game with each state having parameters $(a1, a2, r1, r2, k1, k2, q1, q2, t1, t2, p)$ where k_1, k_2, t_1 and t_2 are the time bounds involved and p represents the index of player with turn on that state. To represent transitions, we created appropriate edges in the graph which changes the current state taking into account switching of turns of players and the changing only the boolean variables affected by that player as each player has a local controller and they cannot affect boolean variables owned by other players in the game. Since we are only looking for admissible strategies we added a common bad state for all the players so that no

one makes any transition to the bad state and creating win-win situation for all the players. The states where time bounds are not satisfied are redirected to the bad state.

Then, we used the algorithm discussed in the previous chapter to find admissible strategies in the above mentioned safety game. The run time observed was quite high because the of the large number of nodes/states involved in the game. The number of total possible states is $2^6 * k^2 * t^2 * 3$. Hence, even for small values for both the time bounds, the numbers of states blows up very fast and hence the overall run time associated with finding the admissible strategies. Approximate run-times for two different cases have been listed below:

k	t	Runtime(approx.)
2	2	15 min
4	2	6 hours

Since, it was infeasible to draw the graph with thousands of nodes even for the lowest time bounds, we simulated it on small examples and tried observing the some of the strategies involved and they came out as expected. Considering an example with $k=4$ and $t=2$, let us take the case when both $a1$ and $a2$ is 1 and rest are 0. The node in this case would be (1, 1, 0, 0, 4, 4, 0, 0, 2, 2, 0). We have two possible nodes where we can go to from this node, (0, 1, 1, 0, 3, 3, 0, 0, 2, 2, 0) and (1, 1, 0, 0, 3, 3, 0, 0, 2, 2, 0). These corresponds to the case where first we set $a1=0$ and $r1=1$ and in the next we keep $a1=1$ and $r1=0$. Clearly, the second transition will get dominated by the first one. We got the same results when we ran the program.

5.2 Burger King Robot Delivery System

This is also a real life application of a Robot which has been given a task of delivering burgers kept on the table motivated from an example in [\[KCJ08\]](#). We may apply this in a franchise like Burger King where one controller can be set up in the desk area and the other



Fig. 5.1: Robot Delivery System

on the robot. The controller has to make sure that when the robot is out for delivery, no more burgers are kept on the table, this may be due to the fact that the robot is configured in such a way that as soon as the burgers are put on the table, it is automatically kept on its hands. Hence, we would like to design a system where the burgers are kept on the table only when the robot is present. And if the robot is not present currently, it must reach the counter in the next step (this next step may take any unit amount of time that can be set by the controllers).

In short, the above problem statement can be represented using Linear Temporal Logic as follows: let a be the action that the burger has to be kept on the table, and b be the action that the robot is out for delivery. So this should be globally true that if the burger is kept on the table, then robot goes out for delivery in the next step. Moreover, if the robot is out for delivery then in the next step, the robot must be back to the counter. This can be represented in LTL as follows:

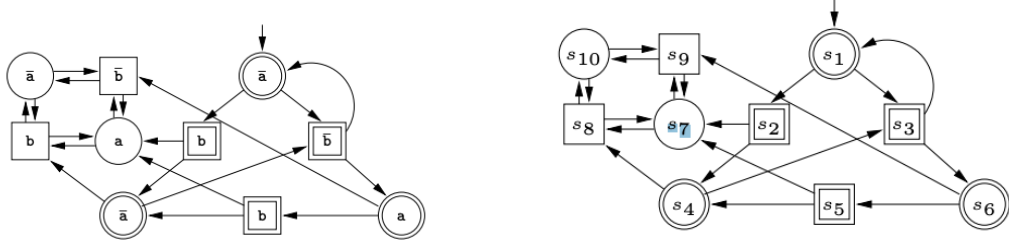


Fig. 5.2: Robot Delivery System represented as an infinite graph game

```

src — -bash — 80x24
users-MacBook-Air:src mayankwadhwani$ python3 robot_delivery.py
The set of dominated strategies:
4 -> 8
6 -> 9
2 -> 7
5 -> 7
users-MacBook-Air:src mayankwadhwani$

```

Fig. 5.3: The set of dominated strategies found in the Robot Delivery System

$$\Box(a \rightarrow \bigcirc b) \wedge \Box(b \rightarrow \bigcirc \neg b)$$

The above LTL was converted to a graph game so that we could find admissible strategies in the same. Fig 5.2 shows the representation of the LTL in a safety game format. The first sub-image shows states with meaningful state names whereas in the second one, the state names have been changed to make it easier for implementation.

Once this is done, the program to find the admissible strategies is run to find the transitions that are dominated by the others. Fig 5.3 shows the results of the program. We can see clearly that the output is as expected. If we remove these edges, then we can assure safety objective for Player 0.

One such dominated strategy that we would look at now is $5 \rightarrow 7$ which maps to the transition $b \rightarrow a$ (this can be verified from the two sub figures in Fig 5.2). If b is true, which means that the robot is out for delivery, then a cannot be true i.e. we cannot put the burger on the desk as this would beat the proper functioning of the system (as discussed above). So the correct strategy to take at such a scenario (when the robot is out) is to not put the burger on the table (which means $\neg a$).

Similar arguments can be given to all the dominated strategies in the figure. And once the set of dominated strategies is found, we automatically get the admissible strategies. Now for the simulation, we start from the initial state s_1 and then depending on the time when burgers are prepared, we move ahead. Since we would only take the admissible strategies, we can be assured that player 0 would always be able to satisfy its safety objective. In an event of more than 1 possible admissible strategies, we can make a choice randomly or give preference to some particular choice (like if we get a choice to deliver or not the burger, we would certainly choose to deliver the burger). In our implementation, we have randomly chosen any strategy since no matter what we choose, we would always end up satisfying our objective.

Hence, this is how we would be able to design a controller for Burger King Robot Delivery System.

Chapter 6

Conclusion and Future Work

6.1 Results

In this paper, we looked at some areas we could make use of admissible strategies. We looked at a few graph games and a few real life applications such as the Scheduler and the Robot Delivery System. We saw how we could design controllers for such systems using admissible strategies where each individual player plays cooperatively to achieve its own objectives.

6.2 Future Work

For the scheduler example, we may make use of parallel programming to improve the efficiency and the run time of the program. We may generate multiple threads to make use of all the cores in the system. This would help us to get the results faster.

We could apply the algorithms and the work to other real life applications as well. We could think of more such examples.

Moreover, in all the examples we have worked on this paper, we have assumed safety objectives for all the players which may not always be the case. There are various other games/objectives that the player may want to meet such as the Büchi or the Muller games.

As a future work, we may expand our controller synthesis to include other types of objectives as well.

Even though admissible strategies was an appropriate way to allow each player meet its own objectives, we may also explore some use some other strategy for the same.

References

- [DF14] Werner Damm and Bernd Finkbeiner. Automatic compositional synthesis of distributed systems. *FM*, 2014.
- [KCJ08] Thomas A. Henzinger Krishnendu Chatterjee and Barbara Jobstmann. Environment assumptions for synthesis. *CONCUR*, 2008.
- [RBS14] Jean-Francois Raskin Romain Brenguier and Mathieu Sassolas. The complexity of admissibility in omega-regular games. *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, July 2014.
- [RBS16] Jean-François Raskin Romain Brenguier and Ocan Sankur. Assume-admissible synthesis. *Acta Informatica*, 2016.

Turnitin Originality Report

Processed on: 18-Apr-2021 17:33 IST

ID: 1562346484

Word Count: 6275

Submitted: 1

BTP Report By MAYANK
WADHWANI

Similarity Index

11%

Similarity by Source

Internet Sources: 9%
Publications: 7%
Student Papers: N/A

2% match ()

<https://hal.inria.fr/hal-01373538>

1% match (Internet from 29-Jul-2014)

<http://152.3.140.5/~abhinath/btp.pdf>

1% match (Internet from 31-Oct-2012)

<http://www.abhimanukumar.com/btpThesis.pdf>

1% match ()

<https://publications.rwth-aachen.de/search?p=id:%22RWTH-CONV-125444%22>

1% match (publications)

[Romain Brenguier, Jean-François Raskin, Mathieu Sassolas. "The complexity of admissibility in Omega-regular games", Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic \(CSL\) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science \(LICS\) - CSL-LICS '14, 2014](#)

1% match (Internet from 05-Nov-2020)

<https://drops.dagstuhl.de/opus/volltexte/lipics-complete/lipics-vol85-concur2017-complete.pdf>

1% match ()

<http://dx.doi.org/10.1109/ACSD.2014.15>

< 1% match ()

<http://livrepository.liverpool.ac.uk/3049696/1/fz.pdf>

< 1% match (publications)

["Implementation and Application of Automata", Springer Science and Business Media LLC, 2007](#)

< 1% match (publications)

[Lecture Notes in Computer Science, 2006.](#)

< 1% match (publications)

["Handbook of Model Checking", Springer Science and Business Media LLC, 2018](#)

< 1% match (publications)

[Lecture Notes in Computer Science, 2003.](#)

< 1% match (publications)

[Grumberg, O.. "When not losing is better than winning: Abstraction and refinement for the full @m-calculus", Information and Computation, 200708](#)

< 1% match (Internet from 03-Mar-2010)

<http://www.nitingupta.org/btp/Senior%20Thesis.pdf>

< 1% match (publications)

[Marcin Jurdziński, Mike Paterson, Uri Zwick. "A deterministic subexponential algorithm for solving parity games", Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm - SODA '06, 2006](#)

< 1% match (Internet from 25-Sep-2019)

<https://www.spiedigitallibrary.org/conference-proceedings-of-spie/10518/105181A/Arbitrary-control-of-the-polarization-state-and-intensity-of-non/10.1117/12.2289225.full>

< 1% match ()

<https://doi.org/10.4230/LIPICS.CONCUR.2019.20>

< 1% match (Internet from 16-Jul-2020)

<https://dspace.mit.edu/bitstream/handle/1721.1/47703/42306186-MIT.pdf?isAllowed=y&sequence=2>

< 1% match (publications)

["FM 2014: Formal Methods", Springer Science and Business Media LLC, 2014](#)

< 1% match (Internet from 16-Nov-2018)

<https://hal.inria.fr/hal-01373538/file/aa.pdf>

< 1% match ()

<http://livrepository.liverpool.ac.uk/3049689/1/frz.pdf>

< 1% match ()

http://publications.rwth-aachen.de/record/696119/files/696119_source.odt

< 1% match (publications)

[Michael Holtmann. "Memory Reduction for Strategies in Infinite Games", Lecture Notes in Computer Science, 2007](#)

< 1% match (publications)

[Lecture Notes in Computer Science, 2016.](#)

< 1% match (publications)

[Lecture Notes in Computer Science, 2015.](#)

< 1% match (publications)

["Foundations of Software Science and Computation Structures", Springer Science and Business Media LLC, 2017](#)

< 1% match (publications)

["Foundations of Software Science and Computation Structures", Springer Science and Business Media LLC, 2014](#)

< 1% match (publications)

[Lecture Notes in Computer Science, 2016.](#)

< 1% match (publications)

[Springer Texts in Business and Economics, 2016.](#)

< 1% match (publications)

[Romain Brenguier, Jean-François Raskin, Ocan Sankur. "Assume-admissible synthesis", Acta Informatica, 2016](#)

Technology by Aman Raj (170101006) Mayank Wadhvani (170101038) under the guidance of Dr. Purandar Bhaduri to the DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI GUWAHATI - 781039, ASSAM 2 CERTIFICATE This is to certify that the work contained in this thesis entitled "Compositional Syn- thesis using Admissible Strategies" is a bonafide work of (Roll No. 170101006 and 170101038), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree. Supervisor: Dr. Purandar Bhaduri Professor Department of Computer Science & Engineering Indian Institute of Technology Guwahati, Assam i ii Acknowledgements We would like to express our gratitude to our supervisor Dr. Purandar Bhaduri for giving us the opportunity to explore this new field of Infinite Games and the much needed zest to delve into some of the state-of-the-art works. iii iv Contents 1 Abstract & Introduction 1.1 Abstract 1.2 Introduction 1.3 Organization of The Report 2 Preliminaries 2.1 Arena 2.2 Controlled Predecessor 2.3 Attractor 2.4 Multiplayer Games 2.5 Reachability Games 2.6 Safety Games 2.7 Strategy 2.8 Winning Strategy 2.9 Dominance for strategies 2.10 Value 3 Methodology 3.1 Algorithms Used 3.1.1 Checking for Winning Strategy 3.1.2 Computing Admissible Strategies v 1 1 2 4 5 5 6 6 7 7 8 8 8 9 9 11 11 11 13 3.2 Our Idea 4 Results 4.1 Example 1 4.2 Example 2 5 Applications 5.1 Scheduler 5.2 Burger King Robot Delivery System 6 Conclusion and Future Work 6.1 Results 6.2 Future Work	
---	--

References 14 16 16 17 19 19 21 25 25 25 27 vi Chapter 1 Abstract & Intro duction 1.1 Abstract Controller Synthesis revolves around the design and implementation of systems working in an environment to satisfy their goals. A popular approach to solve controller synthesis is to find winning strategies in graph games for the system against the environment. Safety games are a class of graph games in which the winning objective for a player is to never visit their corresponding unsafe states. However, in several cases where no winning strategies exist, we find admissible strategies in which players play rationally instead of being adversarial and achieve their goals. Compositional Synthesis refers to the process of realizing a controller for the entire system by breaking it into several controllers which would represent individual components. In our work, we make an attempt to design component-based controllers using Admissible strategies in multiplayer games. 1 1.2 Introduction Games played on graphs with finite vertices have been a topic of study for various years with applications lying in a plethora of examples like controller synthesis. Given, a model of the assumed behaviour of the environment and a system goal, controller synthesis aims at producing a behavioural model for a component that when executing in an environment consistent with the assumptions results in a system that is guaranteed to satisfy the goal. An approach to solve this controller synthesis problem involves graph games. The controller synthesis problem for reactive systems can be modelled into an in- teractive game of strategy typically between two players, Player 0 being the system and Player 1 being its environment. These so-called Graph Games are played infinitely on a finite graph since there are no dead ends. Vertices are partitioned between players and the player owning the vertex decide the next move from that vertex. There are different winning conditions depending on the required specification of the controller. It could be reachability, safety, etc. A strategy followed by a player is the mapping of moves the player makes on each of its vertex. If a strategy allows the Player 0 (the system) to satisfy the required specification or achieve his goals, no matter what strategy the opponent Player 1 (the environment) follows, then it is called a winning strategy for Player 0. Safety Games are a class of graph games which involves safety objectives. The condition in safety games is that Player 0 is not allowed

to enter a set of unsafe/bad vertices, i.e., the whole game play should be confined to a set of safe vertices. To find a [winning strategy for Player 0 in safety games](#), we make [use](#) of its duality with Reachability games, convert the safety objective into reachability objective and implement the algorithm used to find the winning strategy for reachability games. But, there exists many scenarios of safety games where no winning strategy exists for Player 0, in such cases, we look for best possible strategies which are not dominated by the opponent. These strategies are called admissible strategies or non-dominated strategies. 2 Generally, to find a winning strategy, both players play in an adversarial manner, but admissible strategies include those ones in which Player 1 plays rationally or co-operates and focuses on achieving its goal and if possible, without dominating Player 0. This way, we can find strategies which could lead to a win-win situation for both the players. In other words, Player 1 helps Player 0 win as long as he is winning. Compositionality refers to the property that the results observed for the individual components can be used for the entire system. So, if we break the system into several components and find appropriate results for each of them, we would then be able to find the results for the system. Compositional Synthesis helps in realizing the controller for a given system by breaking it into multiple components and obtaining results for each of them. Much work has been done on compositional synthesis by finding the winning/dominant strategies for the players. This is because the winning strategies are compositional i.e. if each component can guarantee to satisfy its objective irrespective of how the other players move, then the entire system would be able to achieve all its objectives. However, this may not always be the case as there may exist scenarios where winning strategies may not exist. In our work, we try to realize component-based controllers using admissible strategies in multiplayer games, i.e. for each player, we would try to find admissible strategies that would be [winning against](#) all [admissible strategies](#) for all [other players](#). We would also [assume that](#) each player may have their own independent objective unlike the case where all players have the same objective as proposed in [DF14]. There are several advantages that we would get because of using admissible strategies. The set of Assume Admissible (AA) profile [is rectangular, i.e. : any combination of AA-winning strategies independently chosen for each player is an AA-winning profile](#). Also, it follows the robustness property. If we replace some [subset of strategies in](#) an [AA-winning profile by](#) other set of arbitrarily admissible strategies, the other players would still be able to satisfy their objectives. Moreover, we would be using the process of iterated elimination of dominant strategies which helps us obtain admissible strategies for the safety games with no winning strategy in the paper.

1.3 Organization of The Report

This chapter introduces the problem statement covered in this report. We provided a brief description of the topic and talked about the importance of controller synthesis in real life. The rest of the chapters are organised as follows: next chapter we discuss all the preliminaries that would help brush up the keywords required. In Chapter 3, we describe the methodology that we are following, we would briefly discuss the algorithms used and the idea that we worked on. In Chapter 4 and 5, we present results as found in various examples and look at some real life applications where our work can be used. And finally in chapter 6, we conclude with some future works.

Chapter 2 Preliminaries

The following section contains a background of the different terminologies that will be used in the later sections. We will be referring to the game described in Fig 4.1.

Fig. 2.1: Example of a Game

2.1 Arena

An arena is defined as a tuple: $A = (V, V_0, V_1, E)$. Here, V represents the total [set of](#) all [vertices in the graph](#). V_0 represents [the set](#) of vertices [for the](#) first player (denoted by Player 0), V_1 denotes the set of vertices for the second player and finally E denotes the set of edges between 2 vertices of the graph.

5 Informally, an arena is a directed graph which contains edges between vertices represent-

ing the different players playing the game. In the example, the q_0 and q_2 will belong to the set V_0 and the remaining would belong to the set V_1 .

2.2 Controlled Predecessor

Given a player i and some set R , the controlled predecessor $CP_{rei}(R)$ of R is defined as: $CP_{rei}(R) = \{v \in V_{1-i} \mid \forall a \in R, (v, a) \in E\} \cup \{v \in V_i \mid \exists a \in R, (v, a) \in E\}$ Informally, the controlled predecessor returns the set of all vertices from which if we start, we are bound to reach some vertex of the given set R . So we break this problem into two, finding all vertices that belong to same set of the player and check if there is any vertex from which we can reach R , if yes, we simply add the vertex to the set. Now, if the vertex doesn't belong to the player's vertices, then we check if all the outgoing edges from this vertex would eventually lead to R . If it does, the second player will have no option but to visit the R .

2.3 Attractor

Given, a Player i

and some set R for a reachability game, the i -attractor given by $\text{Attr}_i(R)$ of set R in arena A is defined inductively using the controller predecessors defined in section 2.2 as follows: $\bullet \text{Attr}_0(R) = R$ $\bullet \text{Attr}_{i+1}(R) = \text{Attr}_i(R) \cup \text{CP}_{\text{rei}}(\text{Attr}_i(R))$ $\bullet \text{Attr}_i(R) = \bigcup_{u \in N} \text{Attr}_i(R)$ While running this process, after some iterations, no new vertex gets added in the Attractor. Generally, after at most $|V|$ steps, the stages of attractor converge and become stationary. Winning region for Player i refers to the set of vertices of the arena from where Player i has a winning strategy. Hence, to get $\text{Attr}_i(R)$ which gives the winning region for the Player i denoted by $W_i(G)$, we need to compute only $\text{Attr}_i(V \setminus R)$.

2.4 Multiplayer Games

A multiplayer game G is defined as a tuple: $G := \langle P, A, (W_i)_{i \in P} \rangle$ Here, P refers to the non-empty set of players involved in the game. A represents the arena of the game as defined in section 2.1. W_i denotes the winning condition for each player i in the set players P . In this paper, we will be focussing on 2-player games. So, $P = \{0, 1\}$. We will be looking at the games from perspective of Player 0 (system). Depending on the objective of the game, the winning condition could be a reachability condition, safety condition, etc.

2.5 Reachability Games

A class of multiplayer games with reachability as its winning condition. The reachability condition $\text{REACH}(R)$ is defined as follows: $\text{REACH}(R) := \{p \in V \mid \text{Occ}(p) \cap R \neq \emptyset\}$ Here, R called the reachability set is a set of vertices such that $R \subseteq V$. V refers to the total vertices of the arena. $V \omega$ denotes various plays or sequences of moves possible in the game. $\text{Occ}(p)$ denotes the vertices reached atleast once in the play. The aim of Player 0 is to reach at least one vertex from a set of vertices R once. Hence, the set of vertices common between $\text{Occ}(p)$ and R should not be null/empty. Player 1 shows adversarial play and tries to avoid him from doing so.

2.6 Safety Games

Safety objective is another type of winning objective involved in multiplayer games. This class of games have a safety condition $\text{SAFE}(S)$ defined as follows: $\text{SAFE}(S) := \{p \in V \mid \text{Occ}(p) \subseteq S\}$ Here, S is a set of safe vertices such that $S \subseteq V$. V refers to the total vertices of the arena. $V \omega$ denotes various plays or sequences of moves possible in the game. $\text{Occ}(p)$ denotes the vertices reached at least once in the play. The aim of Player 0 is to remain confined in S always. In other words, for Player 0 to achieve its objective, at no point in the play, a non-safe or bad vertex from the set $V \setminus S$ should be reached. Hence, the set of vertices in $\text{Occ}(p)$ should be a subset of safe vertices S . Player 1 shows adversarial play and tries to avoid him from doing so. If we observe this game from perspective of Player 1, it can be seen as a reachability game where the objective of Player 1 is to reach any vertex from the set $V \setminus S$. This gives us the sense of duality between these two types of games.

2.7 Strategy

A strategy for a Player i in a game given by the arena $A = (V, V_0, V_1, E)$ is a function $\sigma : V^* \times V_i \rightarrow V$, such that $\sigma(wv) = v'$ where $w \in V^*$ and $v \in V_i$ and $(v, v') \in E$. Hence, σ represents the mapping of states to moves chosen by the Player i against Player $1-i$.

2.8 Winning Strategy

Given, a game $G = (P, A, (W_i)_{i \in P})$, a strategy σ is called a winning strategy for the Player i from a vertex $v \in V$ if every play starting from vertex v following the strategy σ satisfies the winning condition W_i for that player, irrespective of what strategy is followed by the opponent. For safety games, a winning strategy σ for Player 0 from vertex v makes sure that if a play starts from vertex v following that strategy, no unsafe vertex is reached during the play irrespective of what moves are chosen by Player 1.

2.9 Dominance for strategies

Given, a rectangular set of strategy profiles $S = \{ \sigma_i \}_{i \in P}$ where σ_i represents a set of strategies of Player i . A strategy σ is said to very weakly dominate another strategy σ' w.r.t. S , expressed as $\sigma \succ_{\text{weak}} \sigma'$, if from all possible states s , the following condition satisfies: $\forall \tau \in S_{-i}, W_i(\sigma, \tau) \Rightarrow W_i(\sigma', \tau)$ A strategy σ is said to weakly dominate another strategy σ' w.r.t. S , expressed as $\sigma \succ \sigma'$, if the following conditions satisfies: $\bullet \sigma \succ_{\text{weak}} \sigma' \bullet \neg(\sigma' \succ \sigma)$. Here, the strategy σ' is said to be dominated in S as σ dominates it. And a strategy which is not dominated by any other strategy is an admissible strategy in S . To obtain the set of admissible strategies, we iteratively eliminate the dominated strategies.

2.10 Value

After n th step of elimination of dominated strategies, the value of history h for Player i is defined as: $V_i(h) = 1$ if a winning strategy from $\text{last}(h)$, then $V_i(h) = 1$. $V_i(h) = -1$ if \nexists a winning strategy from $\text{last}(h)$ even if the other player helps, then $V_i(h) = -1$. $V_i(h) = 0$ in all other cases. Informally, the value of a state for a player denoted by $\text{val}_i(s)$ is 1 if there is a winning strategy for the player from that state. This means that even if the second player plays in an adversarial fashion, the player will still end up winning the game. Moreover, the value of 0 means that the player will always lose from this state even if the second player plays in favour of the player. For all the remaining cases, we assign the value 0.

Chapter 3 Methodology So far, we have

looked at the basic terminologies that would be used in the rest of the paper. We now turn our attention towards our main idea. We start by briefly discussing the algorithms used in the implementation. This includes algorithms that were used to find winning strategies in safety games and also the algorithm used to find admissible strategies. After that, we briefly comment on the idea that we have worked on.

3.1 Algorithms

Used In this section, we briefly discuss about the algorithms that were employed in our implementation.

3.1.1 Checking for Winning Strategy

We discuss the algorithm 1 in this section which will be used to check if a winning strategy exists from a given initial state for a safety game. We first convert our safety game into its corresponding dual, the reachability game, in the process, all vertices belonging to player 0 in the safety game will now belong to player 1 in the reachability game and similarly for the other player also. As discussed above, player 0 will win the safety game if its dual i.e. player 1 wins the reachability game or in other words

Algorithm 1: Check if winning strategy exists from given state
Input: Arena $A = (V, V_0, V_1, E)$, Initial State, Bad States
Output: True if winning strategy exists from given state
Find Dual of given Input Graph (Assign all vertices for player 1 to player 0 and vice versa);
attractor i = Good States (Initially the given good states act as the Attractor);
while Length of **attractor** i changed from previous iteration **do**
Find CP $rei(\text{attractor } i)$; **Updated Attractor** $i = \text{attractor } i \cap CP\ rei(\text{attractor } i)$; **attractor** $i = \text{Updated Attractor } i$; **end if** initial state not in **attractor** i **then return True; else return False; end**

Algorithm 2: Compute Iteratively Admissible Strategies

Input: Arena $A = (V, V_0, V_1, E)$. The winning conditions of each player (in this case set of bad states for each player)
Output: Set of admissible strategies for the players
while $\exists i \in P, T_{in} i = T_{in} - 1$ **do** **for** $s \in V$ **do** **if** \exists winning strategy for player i from s in graph **then** $V\ alni = 1$; **else if** \nexists winning strategy for player i from s even if the other player helps in graph **then** $V\ alni = -1$; **else** $V\ alni = 0$; **for** $i \in P$ **do** $T_{in} = T_{in} - 1 \cup \{(x, y) \in E \mid x \in V_i \wedge V\ alin(x) > V\ alin(y)\}$ $n = n + 1$; **graph** = **graph** $\setminus T_{in} - 1$ **words** player 0 loses the reachability game. It should be noted that player 0 of the safety game is not the same as the player 0 of the reachability game. Now, to find if the player 0 will win the safety game, player 0 should lose the reachability game or the initial state should not be a part of the final value of **attractor** i . This is because the final value of **attractor** i denotes the winning region for player 0, i.e. the set of all vertices from which we will eventually reach some vertex \in good states and if the initial state is not a part of this winning region, it means player 0 will lose the reachability game which is desired. We now focus our attention towards finding the final value of **attractor** i or the winning region for a reachability game. We can do this by looping till there is no more changes in the **attractor** i and at each iteration finding the controlled predecessor as described above. We take the union of the found controlled predecessor with our **attractor** i and if there is no change break the loop. In this manner, we are able to find if there exists a winning strategy for a player. NOTE: The algorithm described in [RBS14] also requires us to find if from a state we will always lose, that is even if the other players help us, we are still bound to lose. We have in our implementation solved this by a simple idea. We convert all vertices to player 0, i.e. we assume that there is no 'other player' and so if there is only one player, it will not play adversarial to itself. A player will never want to defeat itself. So if a winning strategy exists for the player from a given state, this means that there is still chances of the player to win the game but if no winning strategy exists for this case, we declare that the player will lose the game no matter how optimally it plays.

3.1.2 Computing Admissible Strategies

In this section, we briefly discuss about the second half of our algorithm where we find the admissible strategies for a given safety game. We do this by taking references from [RBS14]. We initialize the T_i set to \emptyset and loop till we find ourselves in a condition where for all states 13 of the graphs, the value of the set T did not change for an iteration, that is the set T has converged. In a particular iteration, we iterate over all the vertices of the graph and compute the values of the state for the player i . We make use of the algorithms as discussed in the previous section. So we can compute whether or not a winning strategy exists or it never exists which corresponds to the values 1 and -1 respectively. If both of these are false, we simply assign the value 0 and continue. We then for all players, iteratively construct the T set, by adding all those edges to this set where the value of start vertex is greater than the value of successor, i.e. all those edges where we will move from a higher value to a lower value. We then remove all the edges present in this T set from the graph as we will never take those paths. We continue this process till the above mentioned condition is reached and then we break the loop. In the end, we have the set of all admissible strategies for the players.

3.2 Our Idea

We

have presented algorithms that help in determining winning strategies and admissible strategies in a given graph game. We wish to expand this idea and try to make use of these algorithms in real life scenarios. We wish to apply them to design controllers which would have cooperative components with an adversarial environment. The individual objectives of each component are satisfied by following admissible strategies (strategies that would allow the environments to be cooperative). In other means, we would try to find a strategy profile $\sigma = (\sigma_1, \dots, \sigma_n)$ such that $G, \sigma, \sigma_e \models \bigwedge_{i \in P} \phi_i$ for all strategies σ_e of the environment. In other words, this means that we have to compute a strategy profile such that each individual strategy is admissible and is winning against other admissible strategies of the other components in the profile and any strategy of the environment. So we would be using admissible strategies to find if the objectives for all the players can be satisfied even if the environment plays in an adversarial fashion. 14 This may lead to a scenario where there are multiple possible strategies possible from a node. In such a case, we would try to think of a best effort strategy, i.e. choose such a strategy that would help us give best results. We would discuss briefly on this in Chapter 5 where we have provided real life applications of this idea. 15 Chapter 4 Results In this section, we present the results that was observed when we run our algorithm on different types of graphs as inputs. These graphs were run to test the functioning of our algorithms.

4.1 Example 1 Fig. 4.1: Example 1 In this example we have $v_3 \in \text{Bad}_0$ and $v_4 \in \text{Bad}_1$ or visiting v_3 will defeat player 0 and visiting v_4 will defeat player 1. It can be seen that there does not exist any winning strategies for the players. So we 16 find the admissible strategies using our algorithm. Our implementation is such that we print the set of edges that is to be deleted from the total set of edges. In other words, we print the edges that we would never visit or the final T set as defined above. It was expected that we should remove the edge $v_2 \rightarrow v_4$. This is because if the player 0 plays this move and reaches v_4 , player 1 will lose the game, and player 1 from this point can act adversarial and move to v_3 which will defeat player 0 also. So player 0 instead will prefer moving to the vertex v_0 . We also expect to remove the edge $v_1 \rightarrow v_3$ since again, if player 1 moves to v_3 and defeats v_3 , player 0 can move to v_4 to defeat player 1. So we expected to remove the edges $v_1 \rightarrow v_3$ and $v_2 \rightarrow v_4$ and our actual results were in accordance with the expectations as can be seen from Fig 4.2. Fig. 4.2: Result for Example 1

4.2 Example 2 In this example we have $v_2 \in \text{Bad}_0$ and $v_3 \in \text{Bad}_1$ i.e., visiting v_2 will defeat player 0 and visiting v_3 will defeat player 1. Again, in this example also, it can be verified that player 0 does not have a winning strategy. We then try to compute the admissible strategies for the same. It can be noticed that the value of q_4 initially for player 1 is -1 since if it reaches q_2 and defeats player 0, player 0 can act adversary and defeat it. But its value from v_0 is 0, since if player 0 moves to q_1 , we can move back to q_0 which will form a cycle of length 2. So since value of q_0 is greater than the value of q_4 for player 1, we are expected to remove this edge. Fig. 4.3: Example 2 In a similar fashion, one can argue to expect $v_1 \rightarrow v_2$ getting removed since if player 1 moves to q_2 and defeats player 0, it can move to q_3 and defeat player 1. So we expected to see two edges $v_0 \rightarrow v_4$ and $v_1 \rightarrow v_2$ getting removed. This is what was observed after running our implementation. Fig. 4.4: Result 2 We can therefore conclude that our implementation of finding admissible strategies for given safety games yields correct outputs. 18 Chapter 5 Applications In this chapter, we shall look at a examples which depicts few real life applications of controllers that could be designed by using admissible strategies. We start by the example of a real time scheduler that is represented as a three player game. Next, we discuss about a robot delivery problem that is represented as a two player game. We represent both these examples as an infinite graph game and find admissible strategies to allow all the players to reach their goals. These admissible strategies are found using the methods discussed in Chapter 3. We shall look at the real time scheduler example now.

5.1 Scheduler This is a real life application of synthesis of a scheduler with two tasks in hand described in [RBS16]. There are three players involved in the system: User, Controller and Scheduler. The overview of the system includes: User can send two different actions a_1 and a_2 to Controller. Upon receiving any action a_i , Controller must issue a corresponding request r_i to Scheduler. The task of Scheduler is to schedule event q_i corresponding to each received request r_i . There are some constraints attached with Controller and Scheduler which they have to meet while fulfilling these requests. On modelling this system into a multiplayer game, each round involves three steps (one 19 for each player) starting with User, then Controller followed by Scheduler. To send an action a_i , User will set it

to true in his turn. Similarly, Scheduler and Controller can set their corresponding boolean variables in their turn to send requests and schedule events respectively. The objective of User is trivial, i.e., accepting for all outcomes. The objective of Controller is to send request r_i within k rounds, after receiving any action a_i . Scheduler has a similar objective, to schedule event q_i within t rounds on getting request r_i . Expressing the requirements of the system in the form of LTL gives us following: $\bullet \varphi_{User} = true \bullet$ $\bullet \varphi_{Controller} = G(a_1 \Rightarrow F \leq k r_1) \wedge G(a_2 \Rightarrow F \leq k r_2) \bullet$ $\bullet \varphi_{Scheduler} = G(r_1 \Rightarrow F \leq t q_1) \wedge G(r_2 \Rightarrow F \leq t q_2)$ Talking about the solution of this game, there exists no winning strategy for both Controller and Scheduler. For any strategy of Controller, the Scheduler can constantly play false without scheduling any event and hence r_i will remain true, hence no more requests can be made by Controller on getting actions. For any strategy of Scheduler, if the time bounds are favourable, Controller can keep on sending requests and Scheduler won't be able to satisfy the requirements. Hence, we look for admissible strategies to satisfy the objectives of all the players in the system. Converting the above mentioned requirements into safety objectives gives us a 3-player safety game with each state having parameters $(a_1, a_2, r_1, r_2, k_1, k_2, q_1, q_2, t_1, t_2, p)$ where k_1, k_2, t_1 and t_2 are the time bounds involved and p represents the index of player with turn on that state. To represent transitions, we created appropriate edges in the graph which changes the current state taking into account switching of turns of players and the changing only the boolean variables affected by that player as each player has a local controller and they cannot affect boolean variables owned by other players in the game. Since we are only looking for admissible strategies we added a common bad state for all the players so that no one makes any transition to the bad state and creating win-win situation for all the players. The states where time bounds are not satisfied are redirected to the bad state. Then, we used the algorithm discussed in the previous chapter to find admissible strategies in the above mentioned safety game. The run time observed was quite high because of the large number of nodes/states involved in the game. The number of total possible states is $2^6 * k_1 * k_2 * t_1 * t_2 * 3$. Hence, even for small values for both the time bounds, the numbers of states blows up very fast and hence the overall run time associated with finding the admissible strategies. Approximate run-times for two different cases have been listed below:

k	t	Runtime(approx.)
2	2	15 min
4	2	6 hours

Since, it was infeasible to draw the graph with thousands of nodes even for the lowest time bounds, we simulated it on small examples and tried observing some of the strategies involved and they came out as expected. Considering an example with $k=4$ and $t=2$, let us take the case when both a_1 and a_2 is 1 and rest are 0. The node in this case would be $(1, 1, 0, 0, 4, 4, 0, 0, 2, 2, 0)$. We have two possible nodes where we can go to from this node, $(0, 1, 1, 0, 3, 3, 0, 0, 2, 2, 0)$ and $(1, 1, 0, 0, 3, 3, 0, 0, 2, 2, 0)$. These correspond to the case where first we set $a_1=0$ and $r_1=1$ and in the next we keep $a_1=1$ and $r_1=0$. Clearly, the second transition will get dominated by the first one. We got the same results when we ran the program.

5.2 Burger King Robot Delivery System

This is also a real life application of a Robot which has been given a task of delivering burgers kept on the table motivated from an example in [KCJ08]. We may apply this in a franchise like Burger King where one controller can be set up in the desk area and the other

Fig. 5.1: Robot Delivery System on the robot.

The controller has to make sure that when the robot is out for delivery, no more burgers are kept on the table, this may be due to the fact that the robot is configured in such a way that as soon as the burgers are put on the table, it is automatically kept on its hands. Hence, we would like to design a system where the burgers are kept on the table only when the robot is present. And if the robot is not present currently, it must reach the counter in the next step (this next step may take any unit amount of time that can be set by the controllers). In short, the above problem statement can be represented using Linear Temporal Logic as follows:

let a be the action that the burger has to be kept on the table, and b be the action that the robot is out for delivery. So this should be globally true that if the burger is kept on the table, then robot goes out for delivery in the next step. Moreover, if the robot is out for delivery then in the next step, the robot must be back to the counter. This can be represented in LTL as follows:

Fig. 5.2: Robot Delivery System represented as an infinite graph game

Fig. 5.3: The set of dominated strategies found in the Robot Delivery System

$\Box(a \rightarrow \bigcirc b) \wedge \Box(b \rightarrow \bigcirc \neg b)$ The above LTL was converted to a graph game so that we could find admissible strategies in the same. Fig 5.2 shows the representation of the LTL in a safety game format. The first sub-image shows states with meaningful state names whereas in the second one, the state names have been

changed to make it easier for implementation. Once this is done, the program to find the admissible strategies is run to find the transitions that are dominated by the others. Fig 5.3 shows the results of the program. We can see clearly that the output is as expected. If we remove these edges, then we can assure safety objective for Player 0. 23 One such dominated strategy that we would look at now is $5 \rightarrow 7$ which maps to the transition $b \rightarrow a$ (this can be verified from the two sub figures in Fig 5.2. If b is true, which means that the robot is out for delivery, then a cannot be true i.e. we cannot put the burger on the desk as this would beat the proper functioning of the system (as discussed above). So the correct strategy to take at such a scenario (when the robot is out) is to not put the burger on the table (which means $\neg a$). Similar arguments can be given to all the dominated strategies in the figure. And once the set of dominated strategies is found, we automatically get the admissible strategies. Now for the simulation, we start from the initial state s_1 and then depending on the time when burgers are prepared, we move ahead. Since we would only take the admissible strategies, we can be assured that player 0 would always be able to satisfy its safety objective. In an event of more than 1 possible admissible strategies, we can make a choice randomly or give preference to some particular choice (like if we get a choice to deliver or not the burger, we would certainly choose to deliver the burger). In our implementation, we have randomly chosen any strategy since no matter what we choose, we would always end up satisfying our objective. Hence, this is how we would be able to design a controller for Burger King Robot Delivery System. 24

[Chapter 6 Conclusion and Future Work](#)

6.1 Results [In this paper, we](#) looked at some areas we could make use of admissible strategies. We looked at a few graph games and a few real life applications such as the Scheduler and the Robot Delivery System. We saw how we could design controllers for such systems using admissible strategies where each individual player plays cooperatively to achieve its own objectives. 6.2 Future Work For the scheduler example, we may make use of parallel programming to improve the efficiency and the run time of the program. We may generate multiple threads to make use of all the cores in the system. This would help us to get the results faster. We could apply the algorithms and the work to other real life applications as well. We could think of more such examples. Moreover, in all the examples we have worked on this paper, we have assumed safety objectives for all the players which may not always be the case. There are various other games/objectives that the player may want to meet such as the Büchi or the Muller games. 25 As a future work, we may expand our controller synthesis to include other types of objectives as well. Even though admissible strategies was an appropriate way to allow each player meet its own objectives, we may also explore some use some other strategy for the same. 26

References [DF14] Werner Damm and Bernd Finkbeiner. Automatic compositional synthesis of distributed systems. FM, 2014. [KCJ08] Thomas A. Henzinger Krishnendu Chatterjee and Barbara Jobstmann. Environment assumptions for synthesis. CONCUR, 2008. [RBS14] Jean-Francois Raskin Romain Brenguier and Mathieu Sassolas. The complexity of admissibility in omega-regular games. Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), July 2014. [RBS16] Jean-François Raskin Romain Brenguier and Ocan Sankur. Assume-admissible synthesis. Acta Informatica, 2016. 27 3 6 7 10 12 17