Optimising Double Elimination Tournaments

Chris Shaw

September 2022

Contents

1 Introduction

Sporting competitions have been a part of human culture for longer than our recorded history. Many sports are played in a pairwise (head-to-head) format, where participants (be they individuals or teams) compete in pairs. These can be physical sports like soccer or tennis, mental sports like chess, and more recently many esports such as League of Legends.

The nature of pairwise competition creates an interesting mathematical challenge when more than two teams are competing. This problem has a large variety of solutions. Some, like the traditional knockout tournament are thousands of years old, while others have only been around for decades or less. The best format for a tournament depends on the desired criteria, as each has its own array of benefits and drawbacks.

Double elimination is a tournament format invented in the 20th century [note to self, find origin source of DET] which has been gaining in popularity, particularly in esport tournaments. Despite this surge in use, there is relatively little academic literature on the topic, and there are a number of open questions pertaining to the format. The aim of this paper is to address some of these questions.

One of the problems we will explore is that of repeat avoidance. At the time of writing this paper, there are a number of bracket hosting websites that are commonly used for running DETs, and each of them has a slightly different algorithm for repeat avoidance. In this paper, we will investigate the optimisation of repeat avoidance algorithms, whether there is a single optimal algorithm, or whether there are a number of equivalently effective solutions.

2 Tournaments as Mathematical Objects

2.1 Notes on motivation and terminology

This paper is intended to appeal to anyone who may be interested in the mathematics of sporting tournament structures, whether their primary background is in mathematics, or the practical uses of these structures in real sports. As such, this paper will explain both perspectives of the problems presented. Mathematicians may see terminology they already understand explained in more simple English, and people with a sporting background will see detailed explanations of concepts they are intuitively familiar with. However, it is only when both of these perspectives are understood together that the true beauty of these structures reveals itself.

It is also worth noting that we will purely be working with mathematical objects here. No properties of any sport will be used, and so any results in this paper can be generalised to all sports, and in fact all forms of pairwise competition. The only assumption made is that the result is binary. That is, there is one winner and one loser in every match. Draws are not allowed, nor is any form of partial result. Often the term "player" will be used to refer to the

competitors, but this player need not be an individual; the term player as used in this paper may also refer to an entire team in a team sport. In some cases, the term player may even refer to a bye, which is a dummy entrant used to pad out empty slots in the tournament.

2.2 Single Elimination Tournaments

Most readers will be familiar with the format of a single elimination tournament (SET), also known as a knockout tournament or knockout bracket. In a SET, each round players are matched in pairs, with the winner advancing to the next round and the loser being eliminated from the tournament. A SET is the simplest form of tournament, and has been in use for thousands of years. It is also the simplest to represent mathematically, being a binary tree with each node representing a match to be played.

In all practical use cases SETs impose the additional requirement that they be represented by a *balanced* binary tree. The formal definition of this term is that for all nodes in the tree, the total number of players in each of its two subtrees must differ by no more than 1.

In professional settings (such as tennis grand slams or the soccer world cup), balanced brackets are often achieved by having qualifying events which limit the number of entrants to an exact power of 2. In other cases, byes are used to fill empty spaces in the first round, and are distributed in such a way to satisfy the formal definition.

This paper will consider *balanced* tournaments with exactly $N = 2^n$ entrants. It is easier to work with a perfect power of 2, and in most cases it makes no difference to the mathematics whether these players are actual competitors or byes.

Below is a diagram showing the basic structure of a SET.

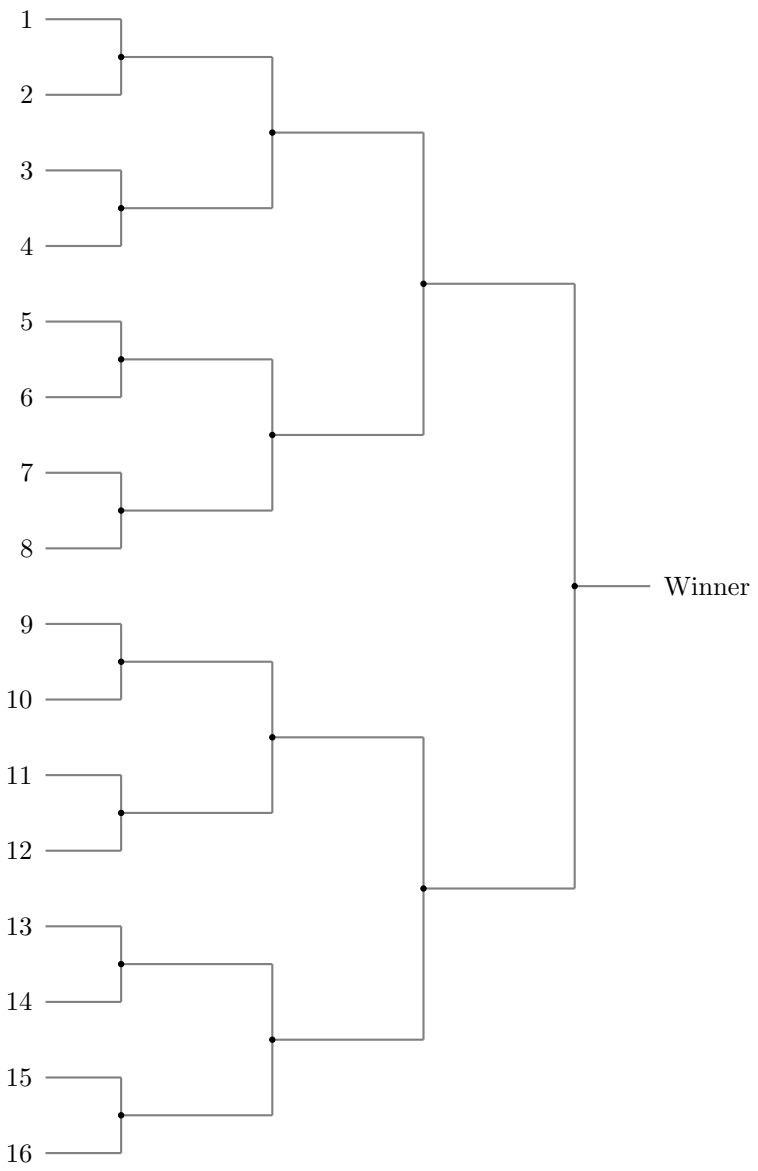

Figure 1: A basic Single Elimination Tournament structure.

2.3 Round and Match Labelling

Next, let's add some labels so that we can refer to specific rounds or matches. In a sporting context, the standard practice is to label rounds in chronological order of play. Round 1 is the first round to be played, and contains all entrants. Next, round two is played, with half the entrants remaining, and so on. Matches, if they are labelled at all, are often given a single sequence of numbers or letters throughout the whole bracket. However, in the context of this paper, it is helpful to use an alternative labelling system so that the mathematics of later chapters is more intuitive.

Rounds will be labelled reverse-chronologically, beginning with the finals as round 0 and ending with the first chronological round being labelled round $n-1$. This allows reference to round i to mean the round with 2^i matches and $2^{(i+1)}$ players. So round 0 has $2^0 = 1$ match, round 1 has $2^1 = 2$ matches, and so on. This labelling is consistent no matter the size of the tournament.

Matches in round i will be labelled with a binary string of length i. This string will start at 0...0 and increment left to right (or top to bottom), with the last match being labelled 1...1. The only match in round 0 is labelled using the empty string, which in this paper will be denoted $\emptyset$.

The purpose of this labelling algorithm is it provides an easy way to locate a given match in the bracket. To do this, start at round 0 (match $\emptyset$). Then take the binary label of the match you wish to find, and parse it left to right. For each number, move to the current position's left child on a 0, or right child on a 1. When the string ends, you will have found your desired match.

Below is a diagram showing labelling of the first four rounds of a tournament using this system.

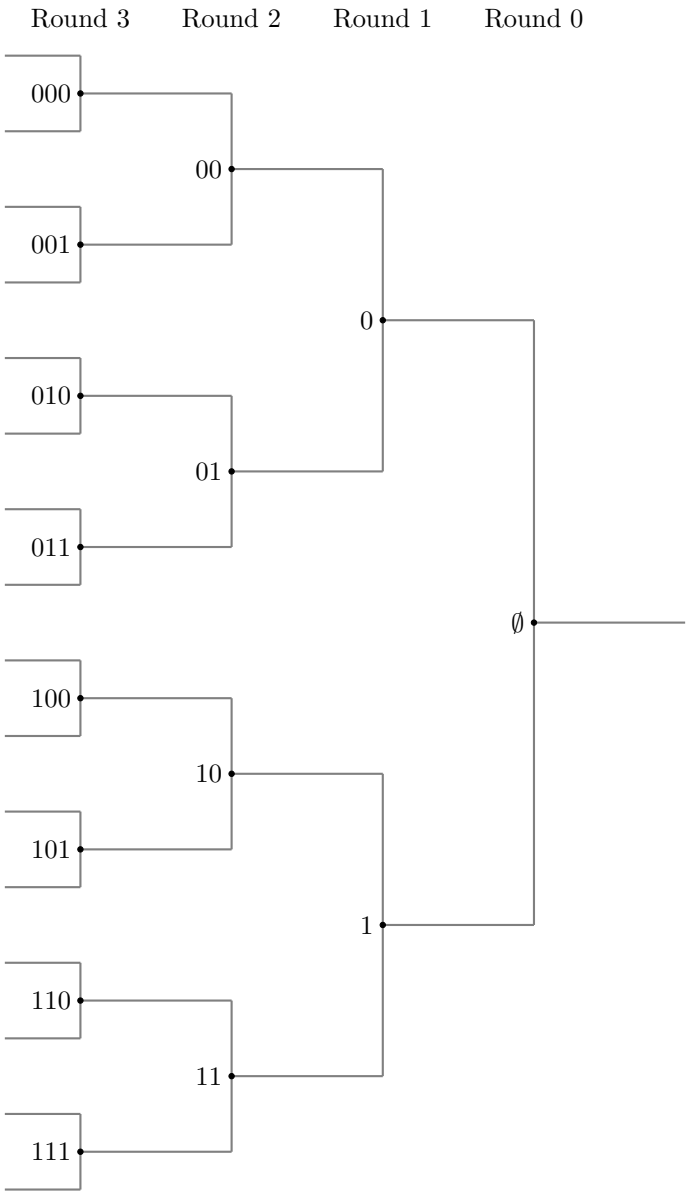

Figure 2: Round and match labelling for a Single Elimination Tournament.

2.4 Double Elimination Tournaments

Double elimination tournaments (DETs) consist of an upper bracket (often called a winners bracket), a lower bracket (often called losers bracket), and a link function which joins them. The upper bracket is where all players begin the tournament, and functions like a SET. When a player loses a match in the upper bracket, instead of being eliminated they are sent to the lower bracket, at a position determined by the link function. The lower bracket functions as a traditional knockout tournament, with winners advancing and losers being eliminated. At the end, the winners of both the upper and lower brackets face off in the grand final. Generally, some advantage will be given to the player coming from the upper bracket to reflect the fact that they have not yet lost a match. For example, the lower bracket player may need to win two matches to win the grand final. The nature of this advantage this varies with each sport using the format, and is not relevant to the mathematics in this paper.

As mentioned, the upper bracket has the same balanced binary tree structure as a SET. The lower bracket is also a binary tree, but the structure of a balanced lower bracket is different to that of a SET. Instead, players are placed further along in the lower bracket depending on how far they progress in the upper bracket before losing. A balanced DET ensures that all players who lose in a single round of the upper bracket are sent to the same round in the lower bracket. The lower bracket then has players alternating between an opponent from the previous round in the lower bracket and an opponent who has just lost in the upper bracket. This paper will only consider balanced DETs.

When labelling rounds of a double elimination tournament, it is helpful to label the lower bracket such that the losers of a given round in the upper bracket are sent to the same round number in the lower bracket. The remaining rounds in the lower bracket are then labelled so that it progresses in 0.5 increments.

Matches in integer rounds in the lower bracket are named by the same method as for the upper bracket. That is, with a binary string of length $\lceil i \rceil$ beginning at 0...0, incrementing left to right (or top to bottom) and ending at 1...1. Matches in non-integer rounds will not need to be labelled.

Below is a visual example of a lower bracket for a balanced DET. Like with a SET, it should be easy to see how this pattern can be extended for arbitrarily large tournaments.

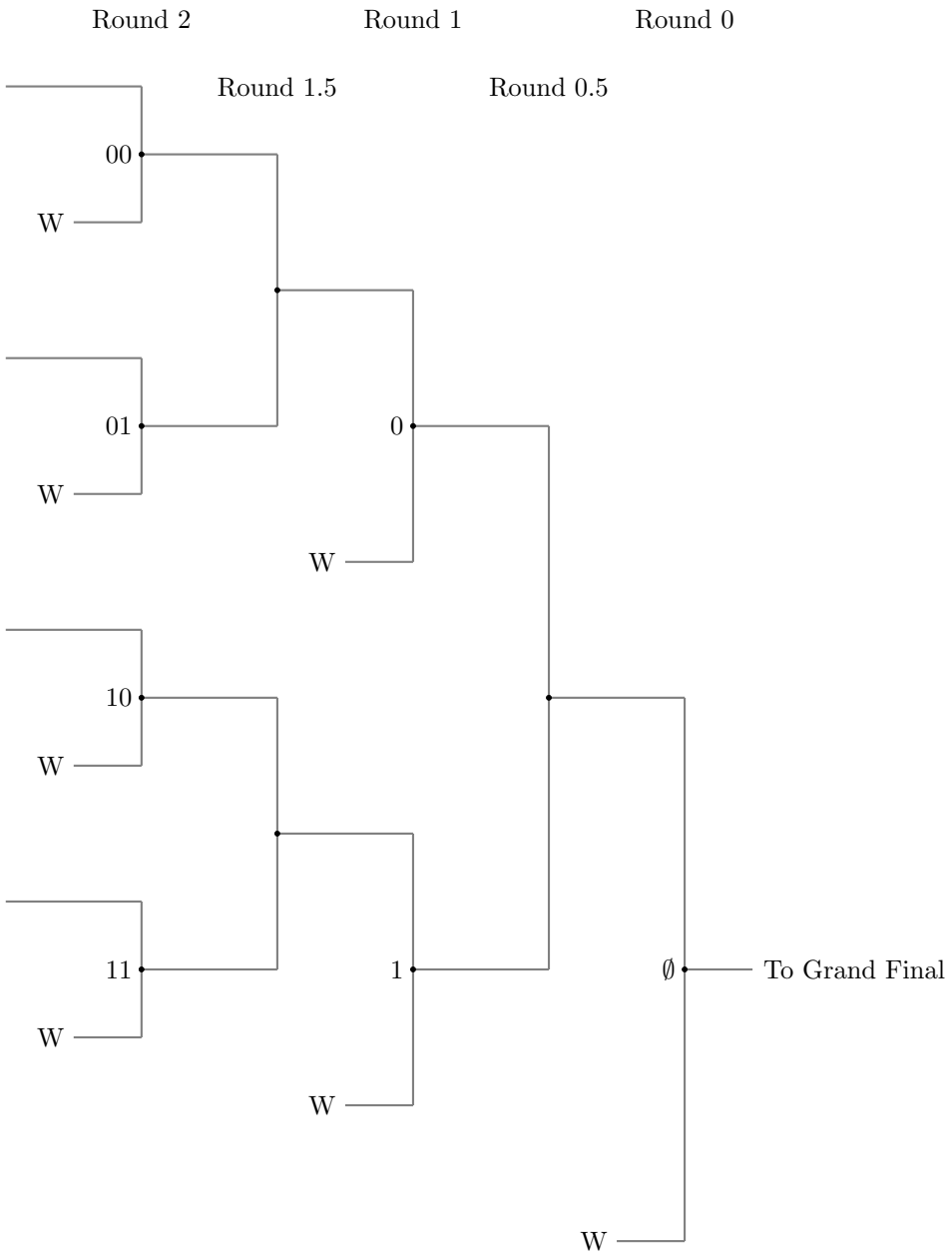

Figure 3: A labelled DET lower bracket. Each 'W' represents a link to the upper bracket.

2.5 Link Functions

Link functions are the main object examined in chapter 1.3, so it is important to define them properly. Some definitions of a double elimination tournament consider a single link function on the whole of the upper and lower brackets. However in practice, link functions for balanced DETs are almost always defined on individual rounds. It therefore makes sense to talk about the link function f_i of a round i, and this will be the language used in this paper.

For a given round i, there are $2^i!$ possible valid link functions in a balanced DET, one for each permutation of the 2^i matches. This number grows incredibly quickly, but if we wish to optimise for repeat avoidance, we only need to focus on a single subset of size 2^i. A proof of this theorem is detailed in Appendix A. For now, I will simply provide the definition that results from the theorem.

Definition ***[Round Link Function]*** *The link function f_i of a given round i is determined by a binary string of length i. For a given match in the upper bracket, the match it is linked to in the lower bracket is determined by a bit-wise exclusive-or operation of its match label and the link function label.*

This definition is equivalent to those which exist in current literature, most notably the one presented in [3]. I have presented it slightly differently so that it is easier to understand. A proof of equivalence is provided in Appendix A.

Like many definitions in this chapter, the link function definition is best understood by visual example, so let's take a look at a few of those.

The first example is the identity link function: $\forall i : f_i = 0...0$

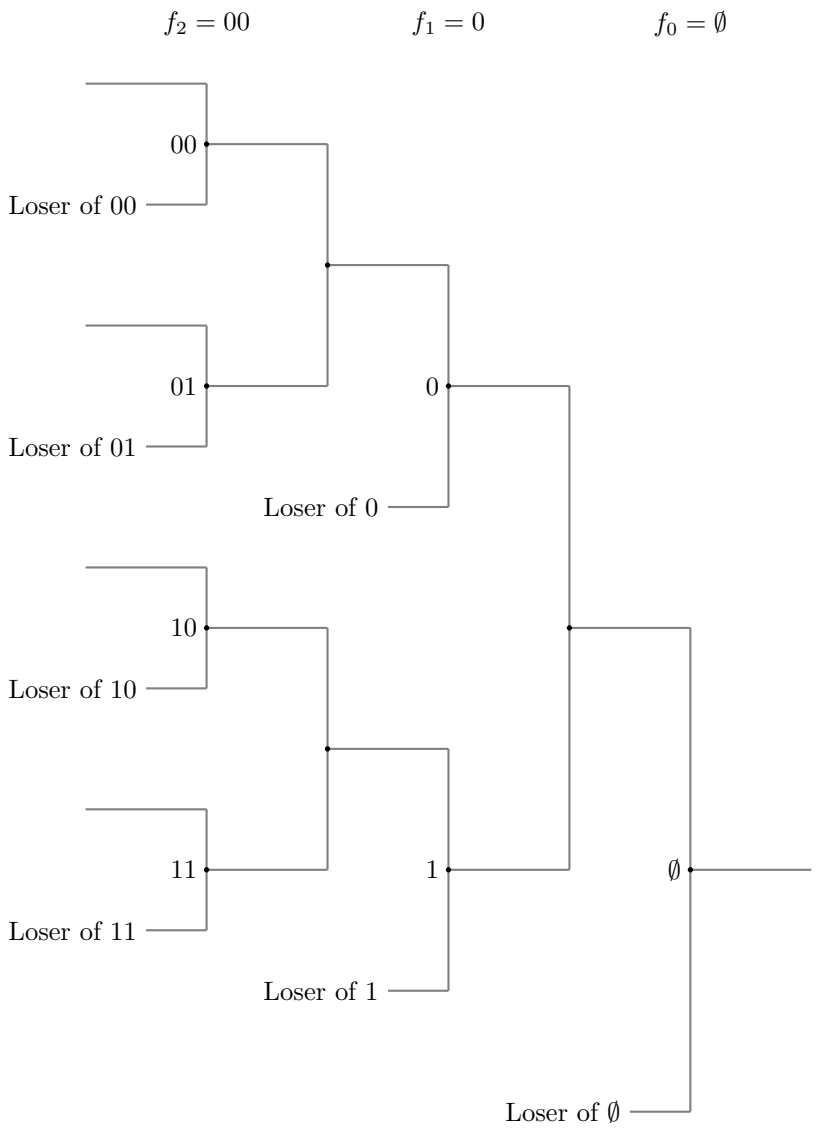

Figure 4: Example 1: The identity link function

As you can see in this example, when $f_i = 0...0$, all matches in the upper bracket connect to the match with the same label in the lower bracket. This is not usually desirable however, so we can use other link functions to shuffle players around in the lower bracket. This can be seen in the next example.

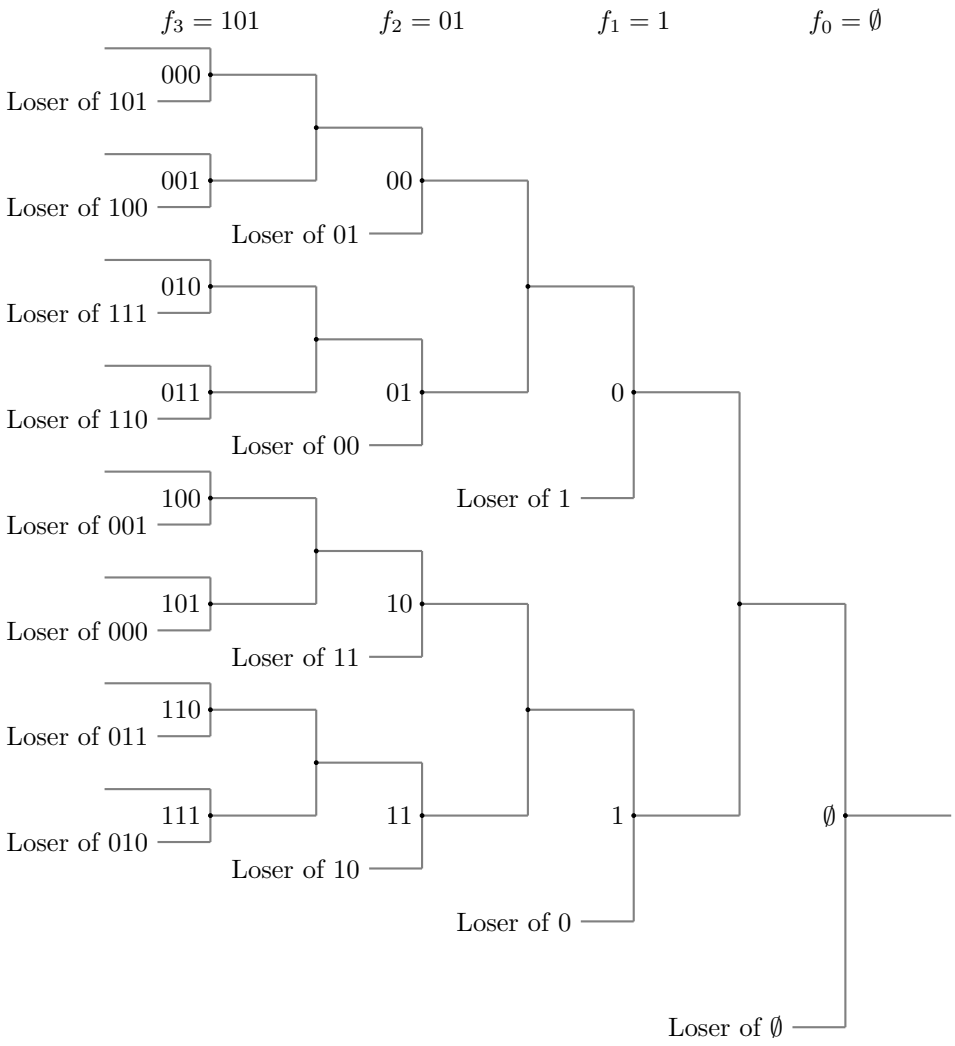

Figure 5: Example 2: A more complicated link function.

Chapter 3 will take a deeper dive into link functions, including the problem of optimising them for repeat avoidance.

2.6 Monotone Rankings

One of the main challenges of working with tournaments is the difficulty of predicting outcomes. A sporting match in a mathematical sense can be thought of as a random variable of unknown probability. Essentially, this presents two layers of uncertainty, not only is the outcome uncertain, but the *probability* of seeing a given outcome is itself unknown. Fortunately, a number of methods of tackling this problem already exist. But before we start looking at solutions, it is helpful to establish some notation:

Definition ***[Match Outcome Probability]*** Let $P(x, y)$ denote the probability that player x defeats player y. In this paper, we will only be considering sports where draws are not possible, therefore $P(x, y)$ is a binary variable. That is: $\forall\, x, y \ : \ 0 \leq P(x, y) \leq 1$ and $P(x, y) + P(y, x) = 1$ as axioms.

Also, while it does not make sense in a real world context, it is helpful to allow the existence of $P(x, x)$ as a mathematical object. By the second axiom above, $\forall\, x : P(x, x) = \frac{1}{2}$. This value makes intuitive sense - if you were somehow able to compete against an exact copy of yourself, you would expect a 50% win rate.

Definition ***[Monotone Rankings]*** A *monotone ranking* is an ordered list of players $a_1 > ... > a_N$ and a probability function $P(x, y) \to [0, 1]$ satisfying the additional axiom that $\forall\, i, j, k \ : \ a_j > a_k \implies P(a_i, a_j) \leq P(a_i, a_k)$.

This is a fairly dense definition, so it is worth spending some time unpacking it. Intuitively, a ranking should list players from most to least skilled. If two players are to face each other, it is reasonable to expect the more skilled player to have a higher chance of winning. While this is not always the case in the real world (rock-paper-scissors scenarios are very plentiful in all kinds of sports), it is very difficult to capture these intricacies in a simple ordered list and so they are assumed away for the sake of simplicity.

The above definition says something subtly different from our intuition of "the more skilled player wins more often." An English translation of the mathematical language would be "Any given player (a_i) has a better chance of defeating a less skilled opponent (a_k) than a more skilled opponent (a_j)."

While it's probably not the first thing that comes to mind, it makes intuitive sense that this condition should be true. It doesn't matter whether you're an amateur, a professional, or anything in between, it's going to be easier to beat me in a tennis match than Roger Federer.

You may ask, what about our previous condition? Surely a ranking should mean the better player wins more often. Actually, that is already covered! It is a fairly simple exercise to show this from our axioms, so I will leave it to you to figure out. (Hint: this is the reason I allowed the existence of $P(x, x)$.)

As I alluded to at the beginning of this section, many such ranking systems that fit this definition already exist. One of the most well known is the Elo rating system [1], which was first used in the 1960s to rank chess players, and is used in a wide variety of sporting competitions today. It is also the foundation of some more modern rating systems, such as the glicko system [2].

2.7 Seeding

Seeding is the practice of arranging the initial configuration of a tournament bracket, and is usually done by first ranking the players by estimated skill level, and then distributing them throughout the bracket according to a particular algorithm.

The practice of seeding dates to the 19th century, when it was first used in tennis tournaments, and has since become commonplace in all sports at all levels. Standard seeding algorithms generally fulfil a number of desirable criteria. In particular, the most skilled players should be evenly distributed throughout the tournament, and should not eliminate each other until the finals. It is also desirable that a stronger ranking should not be a disadvantage with regards to seeding, so as to disincentivise players from manipulating the tournament by hiding or misrepresenting their actual skill level.

With regards to elimination tournaments (SETs and DETs both use the same seeding algorithms), there is a standard method of seeding that is used in almost all events. From here onward, I will refer to it as "standard seeding". It can be defined in a number of ways; personally I like this recursive definition which draws on the match labelling system established in section 2.3:

Definition ***[Standard Seeding Algorithm]*** *Let the seeding of round i be an ordered list of the matches in that round, denoted as s_{i+1}. Each s_i is calculated recursively, as follows:*

There is only one possibility for round 0, so $s_0 = [\emptyset]$. Then, to find s_{n+1}, take two copies of s_n, reverse the second copy, and then append 0 to each of the matches in the first copy and append 1 to each of the matches in the second copy. This gives us the following pattern:

$s_0 = [\emptyset]$
$s_1 = [0, 1]$
$s_2 = [00, 10, 11, 01]$
$s_3 = [000, 100, 110, 010, 011, 111, 101, 001]$

Only one s_i will need to be applied for any given tournament, the one which applies to the chronologically first round. We then seed players by allocating them to matches in that round in the order dictated by s_i. The result will look like the following example:

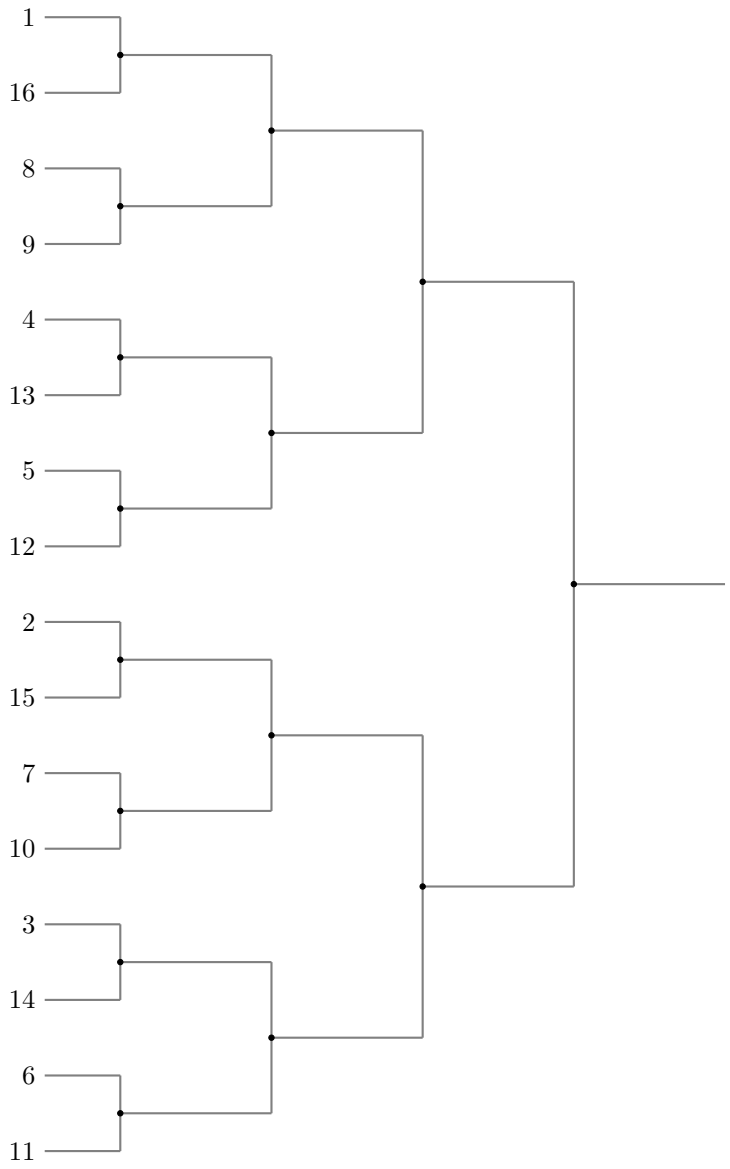

Figure 6: Standard seeding algorithm applied to a tournament of 16 players.

Notice that, barring any upsets (that is, assuming the stronger player wins each match), each round will see the strongest player facing the weakest player, the second strongest facing the second weakest, and so on. In fact, this is the *only* seeding algorithm which satisfies the criteria that a higher seed is always more advantageous than a lower seed when upsets do not occur [citation needed].

But there are some scenarios where it can be argued that a lower seed is more advantageous when looked at in the context of the whole tournament [citation needed].

There is a significant amount of academic literature and case studies on the effects of seeding on the probability of different players winning the overall tournament, particularly for SETs. It is generally considered that the standard seeding algorithm provides some advantage to the stronger players compared to random seeding [citation needed]. This is unsurprising, given the previously stated criteria that a stronger seed should be advantageous to prevent players manipulating the bracket by aiming for a lower seed. The standard seeding algorithm is also considered to provide the best experience for spectators, ensuring that matches become higher skill and more contentious as the tournament progresses, culminating at the final [citation needed].

Though standard seeding forms the basis of almost all non-random seeding algorithms in use today, there are a number of reasons why variations from this standard may be desirable to avoid conflicts. This will be examined in further detail in chapter 1.4.

3 Link Function Optimisation

3.1 The Repeat Avoidance Optimisation Problem

The focus of this chapter is the problem of repeat avoidance in Double Elimination Tournaments (DETs). One of the main advantages of the DET format over the Single Elimination Tournament (SET) format is the increased variety of competition, and the occurrence of repeat matches is contrary to this goal. Therefore, avoiding repeat matches wherever possible in a DET format is generally considered to be desirable. This is done by altering the link function that joins the upper and lower brackets.

Much of the existing academic literature on link functions and the repeat avoidance problem was written by I. Stanton and V. Williams in their 2013 paper [3]. In their paper, they established that it is possible to avoid all repeats in a DET until the final $\log n$ rounds, and that there are many such ways of doing so, but that after this point it is impossible to avoid repeats entirely.

In this chapter, I will take a probabilistic look at the problem, and try to narrow down the space of optimal link functions by imposing stricter criteria on optimality. That is, I will consider the following definition of the problem:

Definition *[Repeat Avoidance Optimisation Problem]* Let $P = a_1 > ... > a_N$ be a monotone ranking, as defined in chapter 1.2.6. Arrange the players of P in a DET bracket according to the standard seeding algorithm defined in chapter 1.2.7. Given a link function $F = f_0, f_1, f_2, ...$ as defined in chapter 1.2.5, it is possible (though in most cases computationally expensive) to calculate the expected number of repeat matches $E(P, F)$. A link function F is said to be *optimal* on P if the value $E(P, F)$ is minimal over all valid link functions.

3.2 Repeat Loops

If we are to investigate repeats in DETs, we should first understand why and how they occur. Take a given link function F, and consider the graph created by the DET using F. While the upper and lower brackets individually have a tree structure, the edges between them created by F cause the overall graph structure to have loops. It is a subset of these loops which determine the possible ways a repeat match can occur.

For a repeat match to occur, first, the players must play in the the upper bracket. The winner will advance to the parent node in the upper bracket, while the loser will move to the lower bracket. The winner will at some point end up either in the grand final, or more likely the lower bracket, having lost in one of the ancestor nodes in the upper bracket. Then, the two players of our original match must both progress through the lower bracket until they meet each other, which is guaranteed to happen if neither of them lose to any other players. At this point, the path taken through the DET graph by the two players forms a loop, which I will refer to as a *repeat loop*.

Any two matches in the upper bracket where one is the ancestor of another uniquely define exactly one repeat loop. Conversely, each repeat loop is defined by exactly two nodes in the upper bracket, where one is an ancestor of the other. Below are some examples of repeat loops. In particular, note the difference that the link function can make to the size of the repeat loop between examples 3 and 4.

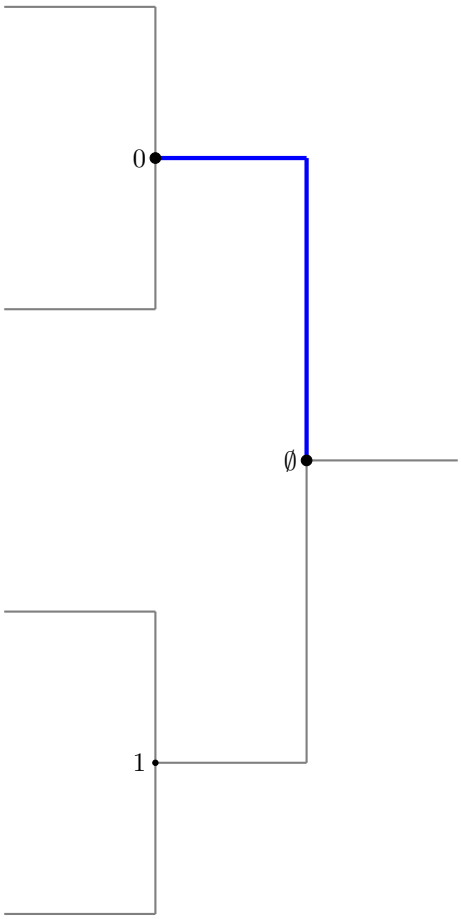

Figure 7: Example 1 Upper Bracket: Repeat loop defined by matches $\emptyset, 0$, with $f_1 = 0$. Loop length = 5.

$f_1 = 0$ $f_0 = \emptyset$

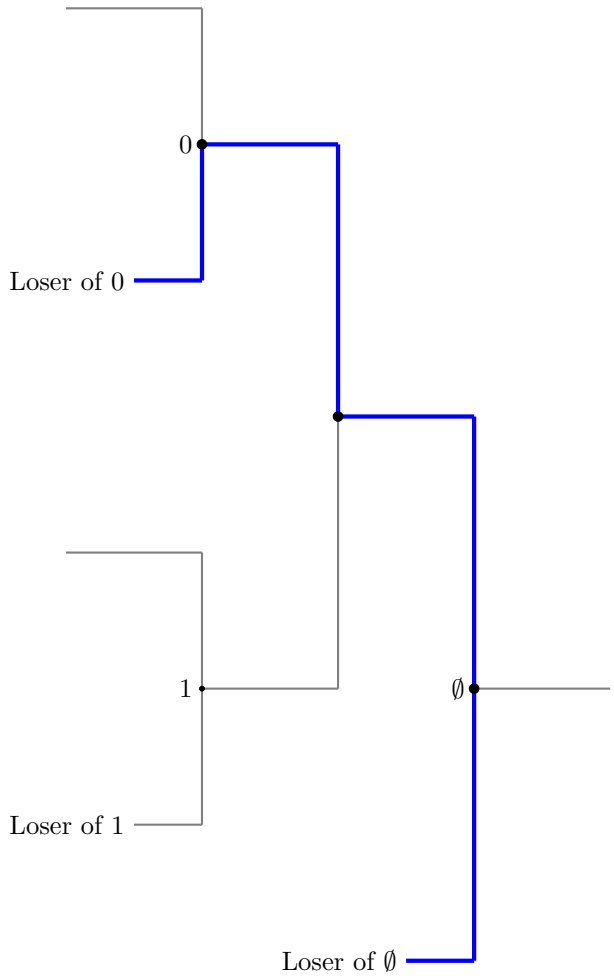

Figure 8: Example 1 Lower Bracket: Repeat loop defined by matches $\emptyset, 0$, with $f_1 = 0$. Loop length = 5.

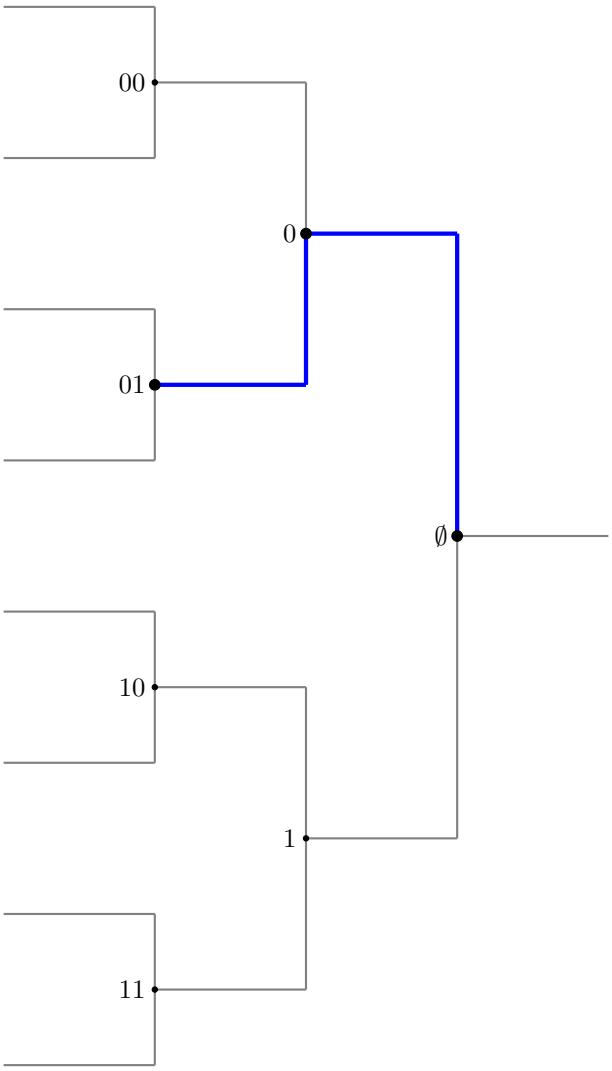

Figure 9: Example 2 Upper Bracket: Repeat loop defined by matches $\emptyset, 01$, with $f_1 = 0, f_2 = 00$. Loop length = 8.

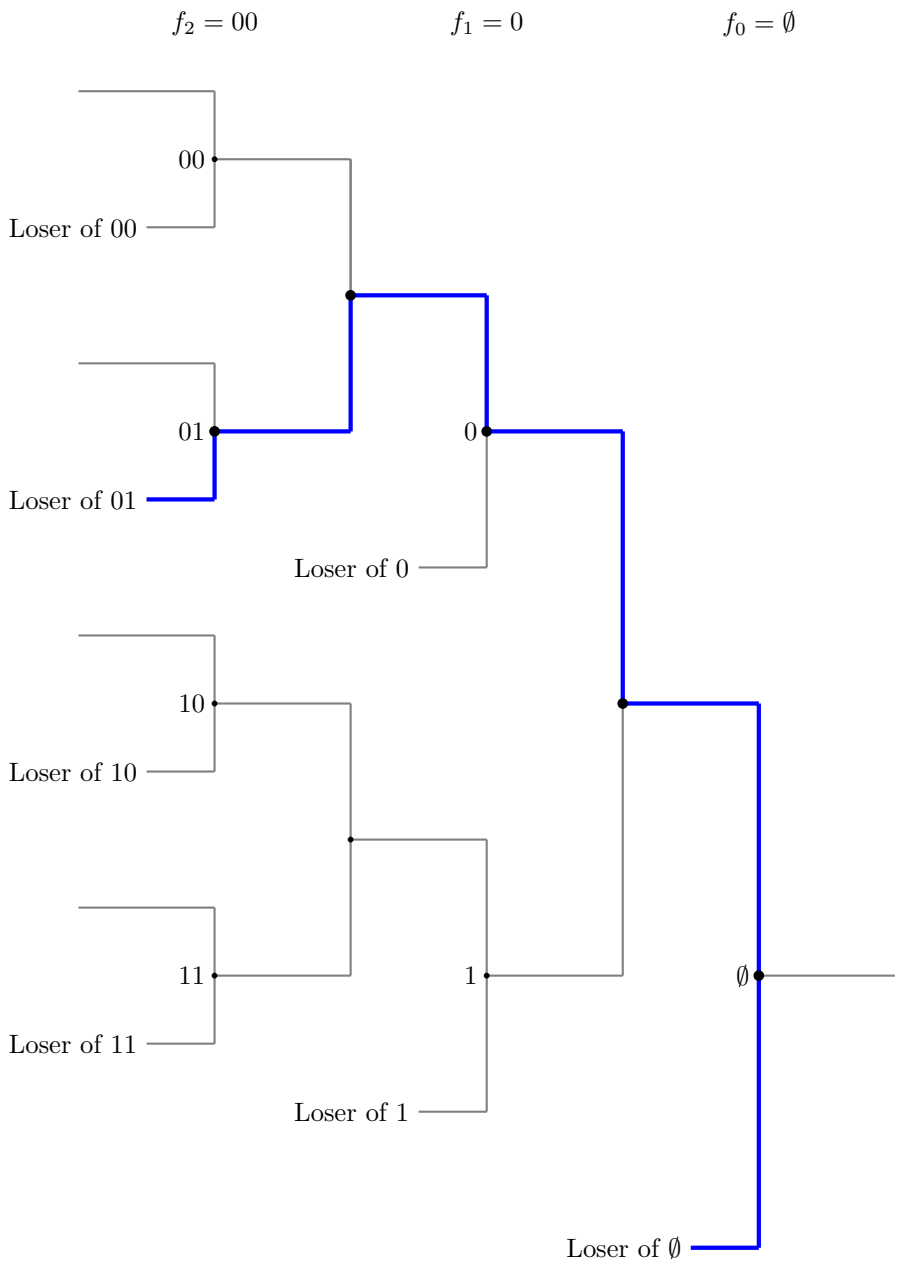

Figure 10: Example 2 Lower Bracket: Repeat loop defined by matches $\emptyset, 01$, with $f_1 = 0, f_2 = 00$. Loop length = 8.

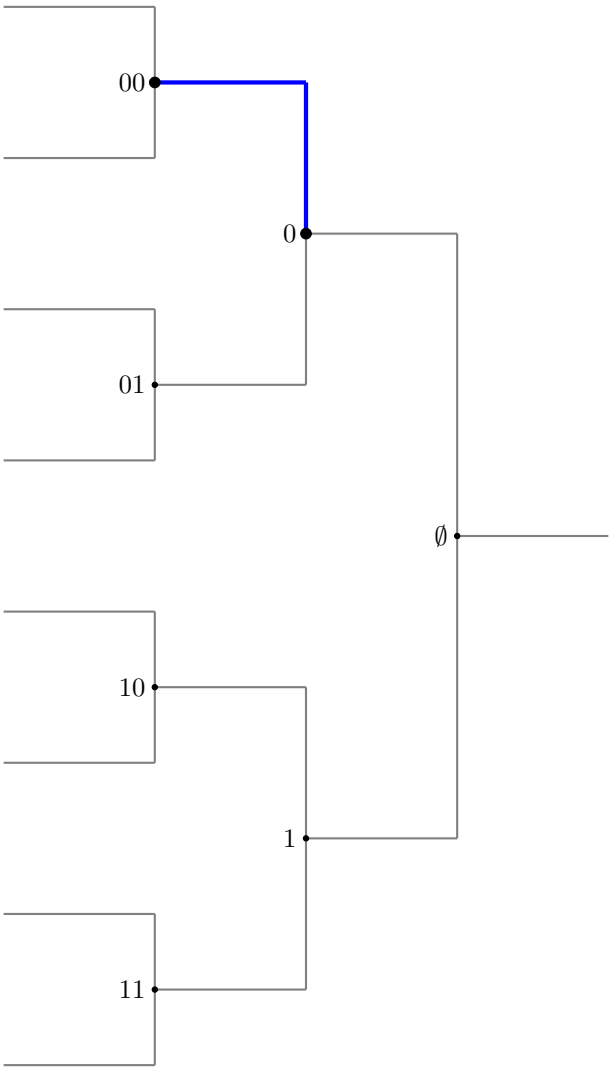

Figure 11: Example 3 Upper Bracket: Repeat loop defined by matches $0, 00$, with $f_1 = 0, f_2 = 00$. Loop length $= 5$.

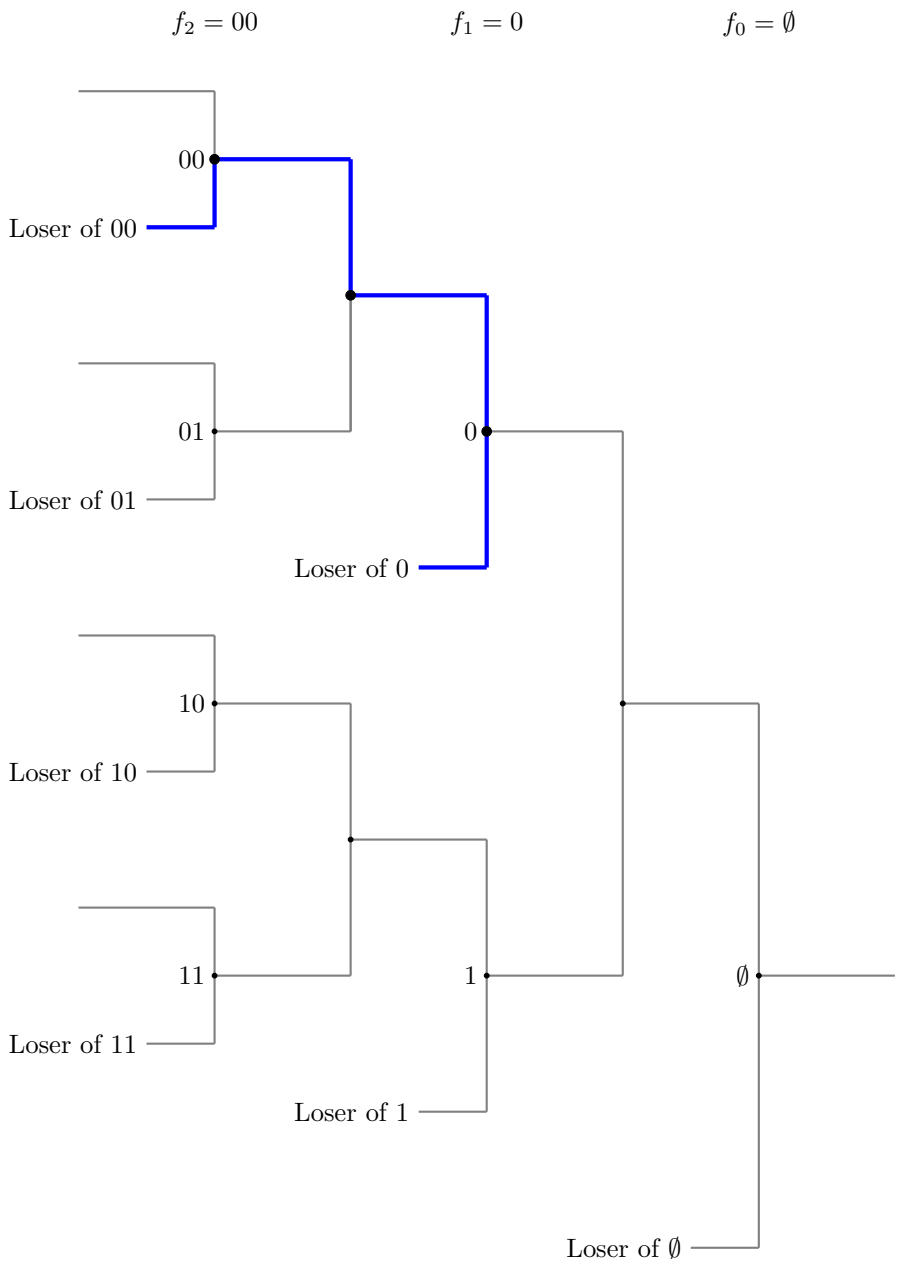

Figure 12: Example 3 Lower Bracket: Repeat loop defined by matches $0, 00$, with $f_1 = 0, f_2 = 00$. Loop length = 5.

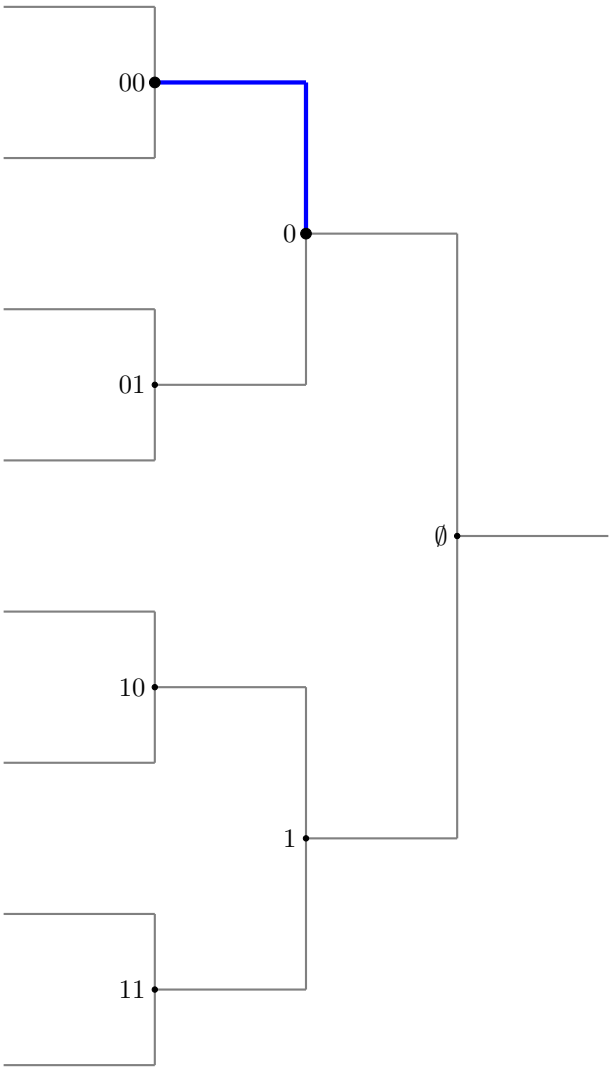

Figure 13: Example 4 Upper Bracket: Repeat loop defined by matches $0, 00$, with $f_1 = 0, f_2 = 10$. Loop length $= 7$.

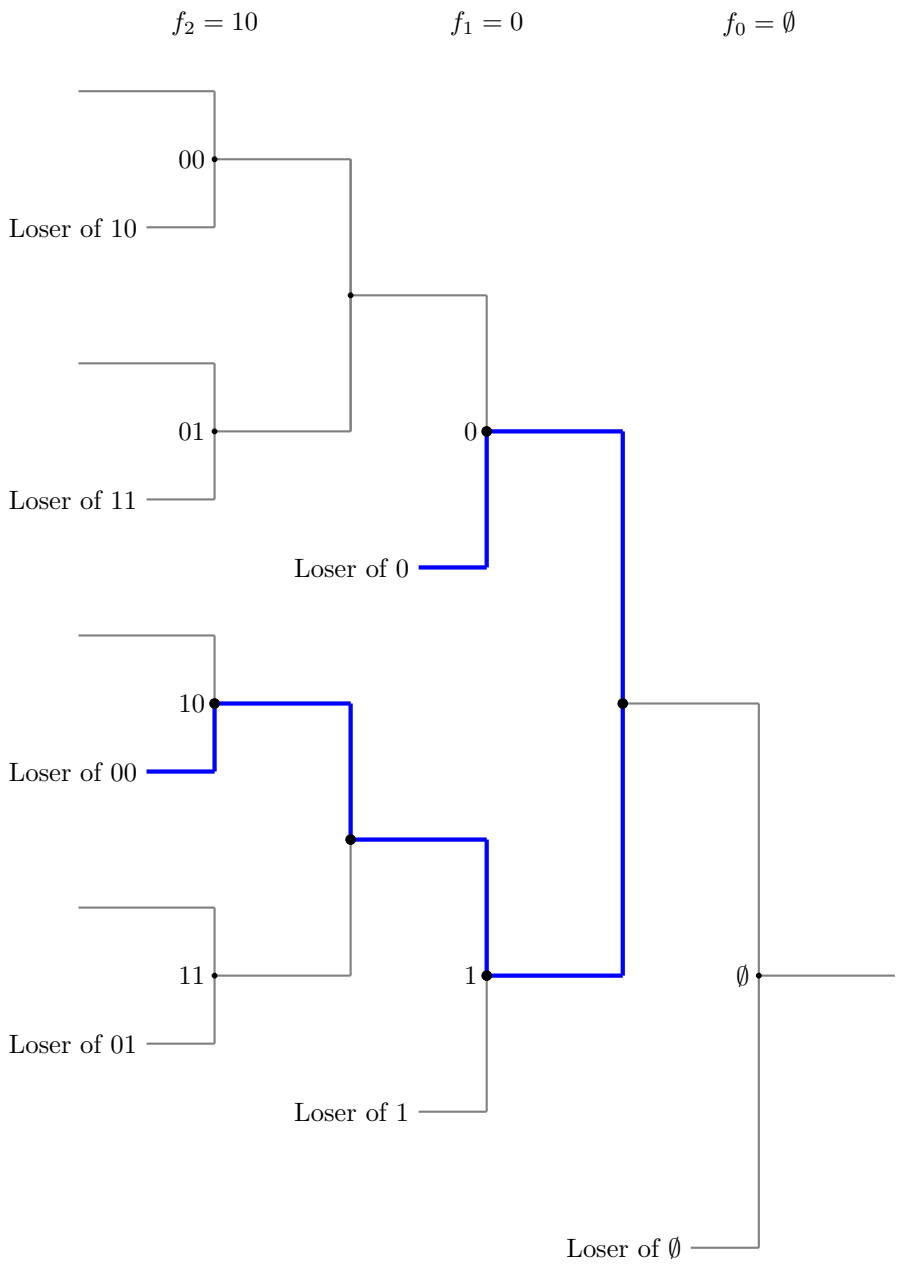

Figure 14: Example 4 Lower Bracket: Repeat loop defined by matches $0, 00$, with $f_1 = 0, f_2 = 10$. Loop length = 7.

3.3 Boundary Cases

Now that we understand how repeat matches occur, let us try to narrow down the problem. The repeat avoidance optimisation problem as defined above is different for any given monotone ranking, so it will be helpful to example just a single ranking. The natural choices for rankings to investigate would be the boundary cases for valid monotone rankings. There are two natural boundary cases to consider for monotone rankings - either the players could be as close to each other in skill as possible, or as far away as possible.

The closest players could be to each other is if all players had exactly the same skill level, and each match had even odds. That is, $\forall\, i, j : P(a_i, a_j) = \frac{1}{2}$. I will refer to this as the *coin flip ranking*, denoted $P_{\frac{1}{2}}$. If, hypothetically, you were to run a coin flipping tournament, this would be an accurate ranking of the players.

For the other boundary case, the farthest away players can be from each other in skill is if more skilled players win over less skilled players with absolute certainty. That is, $\forall\, i, j : a_i > a_j \implies P(a_i, a_j) = 1$. I will refer to this as the *deterministic case*, denoted P_1. There exists a standard definition in academic literature for deterministic probability matrices in tournaments [citation needed], and this is the only monotone ranking which meets this definition.

The deterministic case turns out to be quite trivial. Assuming at least 3 players, there will be a repeat between the 2nd and 3rd seeded players in the final of the lower bracket, and a repeat of the 1st and 2nd seeded players in the grand final. Most link functions will completely avoid all other repeats.

On the other hand, the coin flip case is very interesting and we are able to solve it using our knowledge of repeat loops.

3.4 Solving the Coin Flip Case

If we consider the coin flip monotone ranking, we can calculate the probability of any given repeat occurring. This is because each edge of the repeat loop represents one possible outcome of a match with assumed probability $\frac{1}{2}$. The first two edges which advance the winner and loser can be ignored as which player wins or loses the first match is irrelevant to whether a repeat occurs. For all the other edges, the player has a $\frac{1}{2}$ to either follow or exit the loop. This means the overall chance of a given repeat occurring is exactly $\frac{1}{2^{l-2}} = 2^{2-l}$ where l is the length of the repeat loop. Therefore, it is possible to calculate the expected number of repeats for a link function under the coin flip ranking by simply calculating the size of all repeat loops.

With this established, we can now re-frame the repeat avoidance optimisation problem as one of maximising the lengths of these repeat loops. I also believe that I have found the solution to this problem.

Definition *[Optimal Coin Flip Link Function (OCFLF)]* Define a link function F recursively, where $f_0 = \emptyset$, and f_n is defined to be the link function which minimises the value of $E(P_{\frac{1}{2}}, F)$ given $f_0, ..., f_{n-1}$ are fixed. When all choices of f_n in a particular subtree give the same minimal value, then without

loss of generality pick the leftmost choice (the one that ends in all zeroes).

Before we dive into calculating the OCFLF, let's observe some of its properties. Firstly, all matches in a given round are the same size for a given link function. This is a corollary of Proof 2.5.1 that is presented in Appendix A, and it means that we can just focus on what happens to the first match in each round (label 0...0), as this determines the loop sizes for all matches.

Secondly, the final 1 in f_i can only be at most one position further along than the final 1 of all $f_0, f_1, ..., f_{i-1}$. This is because any swaps that are further down don't interact with the previous terms, and so will be part of a subtree with the same expected number of conflicts.

I wrote an algorithm to calculate the OCFLF, which yields a very interesting integer sequence. Here are the first few terms:

$f_0 = \emptyset$
$f_1 = 0$
$f_2 = 10$
$f_3 = 000$
$f_4 = 0100$
$f_5 = 11000$
$f_6 = 111000$
$f_7 = 1000000$
$f_8 = 10100000$
$f_9 = 011000000$
$f_{10} = 0111000000$

I have uploaded this sequence to the Online Encyclopedia of Integer Sequences, with sequence number A356189.

[If I get time I will draw some more pictures and do a full explanation of the first few steps. I'll need quite a lot of time and pictures though.]

3.5 Open Problems

In this section, I have defined the Repeat Avoidance Optimisation Problem, and made substantial progress at solving one specific case. However, there are still many unanswered questions pertaining to this problem more generally. In this subsection, I will describe some of the problems that remain open.

3.5.1 Patterns in the OCFLF

As mentioned in chapter 1.3.4, the OCFLF is an interesting integer sequence. The fascinating nature of this sequence is not evident from its initial terms, but by the time 50 terms are reached, there is a very observable pattern. Here are the final 10 terms that I have calculated so far, with trailing zeroes omitted:

$f_{90} = 11110011110111111111$
$f_{91} = 1111001111001111101$
$f_{92} = 1111001110011111011$
$f_{93} = 11110011100111110111$
$f_{94} = 1111001110011110111$

$f_{95} = 111100111100111101111$
$f_{96} = 111100111100111110101$
$f_{97} = 1111001111001111101011$
$f_{98} = 1111001111001111011111$
$f_{99} = 111100111101111111111$

It seems that the terms are stabilising to a repeating pattern of the string 111100. It is not obvious that this sequence should show any pattern at all, letalone this pattern in particular.

I conjecture that this pattern would indeed continue indefinitely, since the trailing digits of current terms in the sequence have more influence in determining future terms of the sequence than leading digits.

3.5.2 Well-Definedness of the OCFLF

In the definition of the OCFLF, I mentioned that if all nodes in a single subtree are equally minimal, then the left-most option can be chosen without loss of generality. However, I made no choice as to how to resolve the situation where two separate subtrees are equally minimal. I have manually checked that this does not occur for terms up to and including f_{50}, but it may occur at a later point in the sequence. If it does, the sequence will not be well-defined from that point onward. However, I conjecture that this will not be the case.

3.5.3 Optimality of the OCFLF

In defining the OCFLF, I have made an assumption that each f_i will not change previous terms in the sequence. This is likely to be true, since the number of expected conflicts added by each successive term is roughly half those of the previous term, and I conjecture that this is the case.

3.5.4 The General Repeat Avoidance Optimisation Problem

Much of this chapter has focused on solving one specific instance of the repeat avoidance optimisation problem. However, the problem is different for any choice of monotone ranking, and it remains an open question as to whether the solution is different for each ranking. I conjecture that the OCFLF will be better than most link functions for all monotone rankings, but will not be optimal for all rankings.

4 Conflict Avoidance in Seeding

4.1 Introduction and Motivation

I want to start this chapter by outlining a real world scenario that is the motivation for writing this chapter.

Imagine hosting a national tournament for a particular video game using the DET format. You have hundreds, possibly thousands of entrants from all

around the country. The tournament is open entry, so while the prestige of your event attracts the best players in the country, you also have a significant number of low-medium skill players. Through collaboration with the leaders in each region, you manage to create an approximately accurate monotone ranking of the players, which you intend to use to seed the tournament.

But there is one other constraint that is potentially more important. Each of the main regions in the country hosts their own weekly tournaments. They're coming to your event for a variety of competition, so you want to make sure they don't have to play against players from their own region. This is perhaps less important for players from larger regions where the larger number of entrants makes this difficult, but should be possible for smaller regions.

This problem is actually quite simple for the SET format. There is a single path of progression for each player through the bracket, so you simply need to distribute players from each region evenly. But it is not so simple for a DET. The intricacies of the link function mean that, no matter where a player starts in the upper bracket, they could end up just about anywhere in the lower bracket, depending on which round they lose in the upper bracket. This is notoriously hard to predict, and it is not uncommon for a small number of upsets in the upper bracket to create a large number of region conflicts in the lower bracket.

This is a problem which is faced by tournament organisers around the world on a weekly basis. This is particularly true in fighting game esports, which heavily favour the DET format. At the time of writing, this is a problem which is usually tackled manually, with tournament organisers spending numerous hours trawling through seeding and responding to player requests in order to come up with a desirable solution. The goal of this chapter is to first convert this problem into mathematical language, and then propose more algorithmic solutions.

4.2 Quantifying the Problem

The first step to creating an algorithmic solution to an abstract problem such as this is to apply mathematical language to the problem. The problem we are presented with is essentially a dual optimisation problem. We have two desirable criteria: closeness to initial (skill-based) seeding and avoidance of conflicts. We want to find some algorithmic solution that optimises both of these competing criteria at once, and the algorithm we create will depend heavily on the choices we make in quantifying the criteria.

4.2.1 Skill-Based Seeding

The first criteria is arguably the easier of the two. In the presented scenario, we have a monotone ranking to work with. This means that the optimal solution with regards to skill-based seeding is already known, as discussed in section 2.7. This optimal solution is a sensible baseline from which to start.

Quantifying skill-based seeding then means finding a way of determining how far away a given seeding is from the known optimal solution. This is difficult

to do with an arbitrary monotone ranking, so it is helpful to pick one to work on as an example. For the rest of this chapter, I am going to be using the Elo rating system [1], as this is a very commonly used ranking in various kinds of sports and hopefully most readers should be at least somewhat familiar with it.

Once we have chosen to use the Elo rating system, I propose that a sensible way to measure the distance of a given seeding from the optimal solution is add up the difference in rating between all players that were swapped from the optimal seeding, allowing for swaps only with adjacently ranked players. We are looking to minimise this value, with the optimal solution having a value of 0.

[Will need to insert a few examples here.]

This way of measuring skill-based seeding optimality has a few desirable properties. Firstly, swapping players of similar skill level has much less impact than swapping players with a large skill difference. And secondly, players who end up far away from their optimal position have a much greater impact than those who are close to it, since they will need to be swapped multiple times to get there.

4.2.2 Conflict Avoidance

In the previous section, the second criteria was presented as "avoiding intra-regional matches." However, it is perhaps more helpful to generalise the problem a bit further to "avoiding undesirable matches." In terms of mathematical calculations, it isn't really important why it may be undesirable for two players to play against each other. Perhaps they are close friends/training partners. Perhaps they've played each other at the last 3 tournaments and would prefer new opponents. Or perhaps the are from the same region at a national tournament as in the motivating example.

It may also be the case that some matches may be more undesirable than others. Going back to our national tournament example, if half of the hundreds of entrants are from the host state, and only 10 are from a small region on the other side of the country, it would be far more undesirable for two of the players from the small region to meet in the bracket than it would be for two players from the host state.

With this in mind, the best solution is to create a *conflict matrix* C which assigns a conflict severity between 0 and 1 to each pair of players in the tournament. How this matrix is created will depend on the nature of the conflicts we wish to avoid, and we will work through the national tournament example later in this chapter.

Using the conflict matrix C and the monotone ranking P, it is possible to calculate the number of *expected severity-adjusted conflicts* $E(P, C)$ by taking the total over all conflicts of its severity multiplied by its likelihood of occurring. This is a computationally intensive calculation however, and so the challenge that remains is to set up this calculation in a way that can be done in a reasonable and scalable run-time.

4.3 Match Elo

The Elo rating system [1] defines a method of calculating the probability of a match outcome, given the skill ratings of the two players. However, to solve this problem, we will need to calculate the probability of a player progressing through the whole bracket, without knowing who their opponents will be beyond the initial round.

While it would be possible to calculate this probability exactly using the definition given by the Elo system, it would be computationally expensive to do so. The next best solution is to estimate the probability by assigning an Elo rating to each of the matches. This way, when focusing on a single player we can compare that player's rating to the rating of the match to calculate the probability of the player advancing.

Consider a match between two players A and B with Elo ratings R_A and R_B. The probability of player A winning is given by the Elo system as

$$\frac{1}{1 + 10^{(R_B - R_A)/400}}$$

Or, equivalently, it can be calculated as

$$\frac{Q_A}{Q_A + Q_B}$$

Where $Q_X = 10^{R_X/400}$ is the *win quotient* of X.

We then need to calculate an approximate Elo rating R_W for the winner of A and B. We do this by considering player C who is the opponent for this next match. We need to find R_W so that

$$\frac{Q_C}{Q_C + Q_W} = \frac{Q_C}{Q_C + Q_A} \times \frac{Q_A}{Q_A + Q_B} + \frac{Q_C}{Q_C + Q_B} \times \frac{Q_B}{Q_B + Q_A}$$

Solving the above for Q_W yields

$$Q_W = \frac{(Q_A + Q_B)(Q_A + Q_C)(Q_B + Q_C)}{2Q_AQ_B + Q_AQ_C + Q_BQ_C} - Q_C$$

Remember, in our scenario we are trying to find R_W given players A and B. Unfortunately, the exact answer depends on the rating of the next opponent, so we need to make some assumptions to come up with a sensible estimate for R_W.

Given that C is a player who has progressed further in the bracket, it may be fair to assume that they are a strong player with a high rating. As it turns out, assuming C is as strong as possible causes the above equation to converge to a sensible value.

$$\lim_{Q_C \to \infty} Q_W = \frac{{Q_A}^2 + {Q_B}^2}{Q_A + Q_B}$$

This definition of Q_W (and by extension R_W) has many of the properties we may hope for or expect. When R_A and R_B are close to each other, R_W sits just

above the midpoint, reflecting the roughly 50/50 nature of the outcome. As the distance between the opponent's rating increases, R_W tracks more closely to the higher of the two ratings, reflecting the decreasing probability of an upset occurring.

[I think we can leave it there for now. The next step is to play around in python for a while.]

5 Appendix A

Chapter 2 stated a number of theorems without proof, as the focus of that chapter was on notation rather than proof. The proofs of these theorems are provided in this appendix.

5.1 Proof 2.5.1

In section 2.5, it was stated that only a small subset of possible link functions need be considered. It is worth noting that all practical implementations for DETs currently in use focus on this subset, though until now its optimality has yet to be documented in academic literature.

For a given round i, there are a total of 2^i matches in each of the upper and lower brackets. With no further restrictions, any permutation of these matches would be a valid link function, and there are $2^i!$ such permutations. However, for the purposes of repeat avoidance, the entire link function can be determined by the placement of the first match, for which there are only 2^i possible choices.

Consider the matches $0, 00, 000, ...$ and assume that there is a known optimal configuration for repeat avoidance of these matches. Consider the round link functions $f_0, f_1, f_2, ...$ which match this optimal configuration.

Now consider an arbitrary match a, which can be in any round, and consider its possible repeat clashes with any of its parent matches. Because the same rounds are swapped as in the $0, 00, 000, ...$ case, each of these potential conflicts has the same distance from the original matches. Since this distance is known to be optimal, this configuration is therefore optimal for all matches.

5.2 Proof 2.5.2

In section 2.5, it was mentioned that the link function definition presented in this paper is equivalent to that presented by Stanton and Williams [3]. Here, I provide a formal proof of this claim.

First, it is necessary to provide their definition. It begins with three primitive functions, *swap*, *reverse* and the identity function. Like my link function definition, these primitive functions act on the symmetric group of order 2^n. That is, they take an input of 2^n elements and produce a permutation of them as an output.

Definition ***[Link function primitives]*** *For a string $a_1, ..., a_n$ where n is a power of 2:*

$s(a_1, ..., a_n) = a_{\frac{n}{2}+1}, ..., a_n, a_1, ..., a_{\frac{n}{2}}$
$r(a_1, ..., a_n) = a_{\frac{n}{2}}, ..., a_1, a_n, ..., a_{\frac{n}{2}+1}$
$\varepsilon(a_1, ..., a_n) = a_1, ..., a_n$

As usual, the identity function ε preserves the order of the string entirely. Both of the other functions split the string in half and act on each half individually. The swap function s swaps the two halves, preserving the order, while the reverse function r reverses the order of each half without swapping them.

These functions are then combined in the following way.

Definition ***[Generic link function]*** *Given two strings* $a_1, ..., a_n$ *and* $l \in \{(r, s, \varepsilon)^{\log n}\}$ *where* n *is a power of 2, define a link function:*
if $l = \emptyset, f(a_1, ..., a_n) = a_1, ..., a_n$
if $l[0] = s, f(a_1, ..., a_n, l) = f(a_{\frac{n}{2}+1}, ..., a_n, l[1 : L(l)]), f(a_1, ..., a_{\frac{n}{2}}, l[1 : L(l)])$
if $l[0] = r, f(a_1, ..., a_n, l) = f(a_{\frac{n}{2}}, ..., a_1, l[1 : L(l)]), f(a_n, ..., a_{\frac{n}{2}+1}, l[1 : L(l)])$
if $l[0] = \varepsilon, f(a_1, ..., a_n, l) = f(a_1, ..., a_{\frac{n}{2}}, l[1 : L(l)]), f(a_{\frac{n}{2}+1}, ..., a_n, l[1 : L(l)])$

Here, $l[0]$ is the first character of l, $l[1 : L(l)]$ is the whole string excluding the first character ($L(l)$ referring to the length of l) and $\emptyset$ is the empty string.

Readers familiar with the definition presented in chapter 1 may notice that it is equivalent to the above without reverses. In fact, it is simply a relabelling of ε to 0 and s to 1, presented in a way which I believe to be more intuitive. Therefore, to prove our claim it suffices to show that any string in $\{(s, r, \varepsilon)^n\}$ is equivalent to a string in $\{(s, \varepsilon)^n\}$.

Let $l \in \{(s, r, \varepsilon)^n\}$ be any string. Let l' be the substring of l that begins with the final r in l and continues to the end of l. If we can find a string equivalent to l' that does not contain any instances of r then we can apply this method recursively to l to prove our claim.

Firstly, notice that $f(a_1, ..., a_n, s...s) = a_n, ..., a_1$, that is a string of swaps will reverse the entire input string.

Now, given that the reverse in l' reverses each half of the string, we can equivalently use swaps to reverse the two halves. That is, $r\varepsilon...\varepsilon = \varepsilon s...s$.

Finally, since s is its own inverse, changing ε to s is equivalent to changing s to ε. So we can find an equivalent string to l' by replacing the r with ε and inverting each instance of ε and s.

References

[1] Arpad E Elo. *The rating of chessplayers, past and present.* Arco Pub., 1978.

[2] Mark E Glickman. The glicko system. *Boston University*, 16, 1995.

[3] Isabelle Stanton and Virginia Vassilevska Williams. The structure, efficacy, and manipulation of double-elimination tournaments. *Journal of Quantitative Analysis in Sports*, 9(4):319–335, 2013.