1 Positional games

1.1 Definitions

To define a *positional game*, or *strong game*, or simply *game* from now on, we need a few things.

First of all, we need a "board", V and a collection of "winning subsets", $\mathcal{F} \subset \mathcal{P}(V)$.

The tuple $(V, \mathcal{F})$ constitutes a *hypergraph.* The set V is sometimes called the *vertices* and the set $\mathcal{F}$ is sometimes called the *hyperedges* of the hypergraph $(V, \mathcal{F})$.

Remark We often refer to $(V, \mathcal{F})$ as the game.

The idea is that two players, called *First* and *Second* take turns coloring uncolored vertices of the board.

Initially, the entire board, V starts out with all vertices uncolored.

The object of the game is to be the first to color an entire winning set. The players are named as they are because player First has the benefit of getting the first move.

Note that a vertex which has already been colored cannot be colored again.

The word *play* is meant to represent an instance of a correctly played game from start to finish. Roughly speaking, this means a sequence of partial colorings of V. (TODO: is partial and complete (see below) standard?)

A given point of the play is called a *position* of the board. More precisely, a position is a (partial or complete) two-coloring of V.

Remark Note that a position may not necessarily be attainable when playing by the rules; for instance, I can color one vertex Second without any vertex colored First, which is obviously not a position that can arise in the course of a game played by the rules, but nevertheless a position by definition.

A *draw* happens when the board is fully occupied, yet neither player occupies completely a winning set.

1.1.1 Reverse games

The *reverse* of a given game is obtained if the desired outcome is to avoid occupying completely the winning sets from $\mathcal{F}$.

(TODO: is $reverse(game)$ an operation? i.e. is it possible to define a game from a reverse game? i.e. is it possible to define winning sets for $reverse(game)$, in terms of the winning sets of game? If not, then give counter-example!)

1.1.2 Weak games

In the above definition of a a game, both players (First and Second), strive to occupy the same winning sets, given by $\mathcal{F}$. A player might be interested in settling for a draw. (For instance, if Second knows that he cannot win.)

Thus, we have two players: one player is the *Maker*, and one is the *Breaker*.

We say that Maker wins if he manages to occupy completely one of the winning sets in $\mathcal{F}$, and Breaker wins if he manages to prevent maker.

Note that the notion of who is "first to win" is moot: a player either wins or doesn't win. Note further that a draw is impossible in a Maker-Breaker game.

This "game" is called a the *weak version* of the original game, or the corresponding *Maker-Breaker* game. Note however, that technically this isn't a game at all, according to the above definition: Maker and Breaker have different winning sets! Nevertheless, this is obviously a useful notion.

1.1.3 Reverse weak games

The notion of a *reverse weak game* should now be intuitively clear. The idea is to start with a game, get the corresponding weak game and then reverse that. However, since a weak game is technically a not a game, we should give an explicit definition of what reverse means in reference to a weak game.

Suppose we have a game with the hypergraph $(V, \mathcal{F})$. The corresponding weak game as a Maker and a Breaker. The reverse weak game has a player trying to avoid making, and a player trying to avoid breaking. That is: we have an *Avoider* and an *Enforcer*.

Avoider tries to avoid occupying completely a winnig set from $\mathcal{F}$, and Enforcer tries to prevent Avoider from doing so, that is, tries to enforce Avoider into occupying completely a winning set.

(TODO: Is a reverse weak game a weak game? if not, then give a counter-example)

1.2 Library interface

(TODO: This section)

1.3 The game tree

A good conceptual tool when reasoning about positional games is the so-called game tree, corresponding to a given game.

The root node of the game tree is the starting position of the game. That is to say, the empty board.

The children of the root node is all possible positions of the board after First has made his move. The children of *those* nodes are the position after Second has made his move, and so on.

The game tree clearly finite, but can be very large.

TODO: Insert example of game tree here

2 Optimal play

2.1 Introduction

In this section, we will introduce an algorithm called the minimax algorithm which can play any positional game (and more genral games) in an optimal manner.

Quite clearly, any such claim is bound to be about an algorithm with performance issues. There are common optimizations, such as alpha-beta pruning, which we also talk about.

The natural structure on which algorithmic play takes place is the game tree, as introduced in section **??**. A playing algorithm can then be seen as a search algorithm on the game tree.

2.2 Optimal play

What do we mean by "optimal play"?

In this section, we will give some pretty intuitive neccesary conditions.

Clearly, a player cannot play optimally if he squanders an oppurtinity to win. More precisely; if current position is a win for the moving player, then he must make a choice which is also a win for him.

Furthermore, if the moving player does not have any opportunity to win in the given position (i.e. none of the leaf-nodes in the tree attached to the current node contains a winning node for the moving player), but if he does have an opportinity to tie, he must still have an opportunity to tie after he makes his move.

Thus, if a player can play as above, then since the game can't go on forever, he will win if he can win, and if he can't win but he can tie, he will do that.

TODO: Standard definition of optimal play?

2.3 The minimax algorithm

In this section, we will see how the considerations in the previous sections guide us to a pretty intuitive algorithm which leads to optimal play.

It is intuitively clear that it is always possible to play as described in the previous section.

The key in order to find an explicit algorithm is to extend the notion of First playwe win. The notion of a winning position is connected to the definition of a game, but not to how the player will play i.e. his strategy. Thus, the idea of a winning node is restricted to the leaf nodes. We will use the notion of optimal play in order to extend this definition to non-leaf nodes.

In terms of the game tree, what does it mean for a given position to be a First player win? The definition is inductive.

In the base case, i.e. we have a leaf node, the definition of winning node is clear from the rules of the game. If we are not on a leaf node, we break the definition up into two cases:

- Case 1: It is Firsts turn to move. In this case, the position is a First win it has a child which is a first win.

- Case 2: It is Seconds turn to move. In this case, the position is a First win if, for all of it's children, they have children which is a First win.

The notion of a position being a Second win is defined similarly.

It is easy to see that if a position is a First win, it cannot be a Second win, and vice-versa. This does not mean that a position must be either a First win or a Second win; and we call such positions Neither win positions.

We have now defined a coloring of any game tree, with three different colors: First win, Second win and Neither win.

The following results follow directly from the definition of optimal play:

If the game is in a First win position, then the play won't change color if First plays optimally. (And similarly for Second.)

If the game is in a Neither win position and First plays optimally, then the game will never be in a Second win color, but it might turn into a First win color unless Second also plays optimally. (And similarly for Second.)

Finally: if both players play optimally, then the color of the game won't change.

From the above results, the central theorem, which corresponds to the so-called minimax algorithm follows easily: The color of a node is the same as the color of the leaf node which results when both players play optimally. (This is well defined because of the above corollary.)

To get something a bit more operational, we define an ordering on the set of colors:

$$\text{Second win} < \text{Neither win} < \text{First win}$$

It is clear that First plays optimally if he at all times makes choices with maximum color. Similarly, Second plays optimally if he at all times makes choices with minimum color.

Thus, to find the color of a position in which it's Firsts time to move, we look at each of it's children in order. If any child is a First win, we know that our node is a First win, and can stop searching.

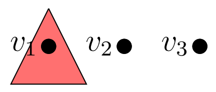
To find the color of a position in which it's Seconds time to move, we look at each of it's children in order. If any child is a Second win, we know that the our node is a Second win, and can stop searching.

This leads to the following mutually recursive definition of color for a node, which can me used to play optimally:

```
winner play@(Play First _) =
  max $ map winner $ options play
winner play@(Play Second _) =
  min $ map winner $ options play
```

Because Haskell is a lazy language, it might *not* need to calculate the winner function for all of it's children, as desribed above. In fact, it might not even have to evaluate all options in order to know who is the winner.

3 Hypergraphs



References

József Beck, Combinatorial Games: Tic-Tac-Toe Theory *Cambridge University Press, 2008*