Dependent Type implementation for Connect Four

Zi Chen

1 Introduction

Board games and economical strategies have the same goal in which the optimum steps are picked in order to increase the reward once completed. This general concept is known as Game Theory where the aim is to maximise the profit for a player or to minimise loss/risk by choosing optimal strategies. *Payoffs* are used to determine how much a player receives when they achieve a win. These payoffs can be positive or negative and do not necessarily correspond to an amount won. The numbers can indicate points, prizes or profit.

Various games can be categorised into several types depending on the characteristics of the game [1]. The types are described as follows:

- *Cooperative*, *Competitive* or *Hybrid Games*. The game is known to be cooperative if players can collaborate and communicate within the game. Otherwise, they are competitive with players competing against one another and Hybrid games holds elements of both types.

- *Zero-Sum* or *Non Zero-Sum* games. In zero-sum games, the overall payoff is divided among the number of players meaning that the amount won by players is exactly the amount lost by others. Otherwise, they're non zero-sum games.

- *Simultaneous* or *Sequential* games. If actions from all players occur at the same time/turn, the games are known as simultaneous games, an example is rock-paper-scissors. Actions taken based on prior information of another player's previous moves is known as sequential games. These are games such as Tic-Tac-Toe, checkers, and chess.

- *Symmetric* or *Asymmetric*. Games can vary depending on the choices the players must make, if they have the same choices as all other players then it is a symmetric game. Otherwise, the game is asymmetric and infers that the player plays a different role causing a difference in their payoffs.

The number of players can also vary depending on the game. *Coalitions*, where players can form alliances to work together, can also benefit players in some scenarios and games.

1.1 Higher order games

The focus of higher-order games is centralised on sequential and zero-sum games where the goal is to maximise the player's playoff by using extensive form. *Extensive form* [7] uses trees to represent the ongoing choices within a game and represents the sequential games. Whereas *normal form* uses matrices for simultaneous games. The outcome type for higher-order game can be represented by a arbitrary type R with X corresponding to the move set viable for the player. Each player's goal can be defined by *quantifiers* which are of the type $(X \to R) \to R$.

Following from the notation in [4], the types $\Pi(x : X), A(x)$ and $\Sigma(x : X), Ax$ from HoTT are written as $(x : X) \to A(x)$ using functions and $(x : X) \times A(x)$ using dependent pairs respectively. This is so that they can be implemented and proven by using Agda and provides a way to model the games. Thus, we can use this notation to define games such as Tic-Tac-Toe, Connect 4 and other similar sequential games. For example, Tic-Tac-Toe is modelled by the linearly ordered set $-1 < 0 < 1$ where each is the outcome for the game. 1 denotes the first players win, -1 as the second player winning and 0 if it is a draw [3].

So, the two players from Tic-Tac-Toe can be shown as the quantifiers of max,min: $(X \to R) \to R$. Generally this can be adapted to other games and expanded upon to for more complex sequential games with perfect information (where all choices in the game are known by the players). The formalisation of higher-order games using dependent type theory is defined in the next section with Connect 4 as an alternative example to the Tic-Tac-Toe in the paper [4].

2 Dependent type theory and Implementation

When analysing sequential games, the plays that players can make are based upon previous choices which can be represented by the form $x_0, ..., x_{n-1}$. This enables the definition of dependent type trees using induction with the conventions of X ranging over types, X^t ranging over type trees and X^f ranging over type forests.

Definition 2.1. (Dependent Type Trees)

1. The initial empty tree is denoted by [ ].
2. For each X of a type with $X^f : X \to \text{Tree}$ as a family of trees that's indexed by X. Then $X :: X^f$ is a type tree which denotes that X is the root of the tree with subtrees $X^f x$ for each $x : X$.

The use of the function $\_ :: \_$ in the definition of dependent type trees is similar to the concatenation of lists in Agda but has the type $(X : \text{Type}) \times (X \to \text{Tree}) \to \text{Tree}$ instead.

Using Connect 4 as an example, the moves possible at each turn are used to form a dependent type tree with the root being the type of the 7 possible columns that you can put a piece into to fill the columns. When one of the columns get full then the next players choices are reduced by one unless a win has already occurred. If the game continues until all columns are filled, then the next player has no more moves so ends on an empty sub-tree meaning the game is a draw.

The sequence $x_0, ..., x_{n-1}$ denotes all previous moves of each player mentioned before. This is just a path within the dependent type tree that has just been defined. So we can define this path formally.

Definition 2.2. (Paths) *Paths* can be defined on dependent type trees X^t similarly as before through the use of induction on X^t.

1. The empty type tree has a singular path defined as $\langle\rangle$.

2. For each X of a type with $X^f : X \to \text{Tree}$. Then every $x : X$ and every path which can be denoted as $\overrightarrow{x}$ in $X^f x$, there exists a path $x :: \overrightarrow{x}$(x is the head and $\overrightarrow{x}$ is the tail) from the tree $X :: X^f$.

This results in the initial path having a type as $\langle\rangle : \text{Path}([])$ and longer paths as $\_ :: \_ : (x : X) \times \text{Path}(X^f x) \to \text{Path}(X :: X^f)$. The paths from the trees are the sequence of legal moves viable to each player.

Going back to higher order games, the moves at a given point in the game is dependent on the prior moves made and the notion as a quantifier is required for the definition of higher order games. For Connect 4, we can take $R = -1, 01$ but R can vary depending on the game. Similarly in [4], let KX be an abbreviation for $(X \to R) \to R$ where R can vary but fixed based upon the use. The functions with the type of KX are known as quantifiers.

Min, max: $(X \to R) \to R$ are the simplest examples of quantifiers with the idea of the set of outcomes R being the linearly ordered set. For Connect 4, this set will be $-1 < 0 < 1$. So through available moves X and given a local outcome function $X \to R$, the min/max quantifier will generate a preferred outcome. We use the quantifier to define a quantifier tree.

Definition 2.3. (Quantifier Tree) Dependent type trees were defined previously so they can be expanded to become *quantifier trees* by using the same inductive approach.

1. For the empty tree that was denoted as [] before, the quantifier tree is denoted by type $\mathcal{K}([])$ with $\langle\rangle$ as the only element.

2. Then for type trees represented by $X :: X^f$, it's quantifier tree equivalent has root KX with a branches of $\mathcal{K}(Xfx)$ for every $x : X$.

Following the definition, the empty quantifier tree type is $\langle\rangle : \mathcal{K}([])$ and larger trees as $\_ :: \_ : KX \times ((x : X) \to \mathcal{K}(X^f x)) \to \mathcal{K}(X :: X^f)$. Furthermore the variable $\phi : K(X)$ ranges over quantifiers. Similarly, $\phi^t : K(X^t)$ for quantifier trees and $\phi^f : (x : X) \to \mathcal{K}(X^f x)$ for quantifier forests.

Products of quantifiers [2] can also be defined from given quantifiers $\phi \in KX$ and $\gamma \in KY$ by

$$\phi \otimes \gamma \in K(X \times Y).$$

Which can be extended to define binary dependent product of quantifiers.

Definition 2.4. (Binary Quantifier Products) For a quantifier $\phi : KX$ on type X, a family Y of trees indexed by X and $\gamma : (x : X) \to K(Yx)$ which is a family of quantifiers. The new quantifier $\phi \otimes^K \gamma : K((x : X) \times Yx)$ by

$$\phi \otimes^K \gamma(q) := \phi(\lambda x.\gamma(x)(\lambda y.q(x, y))).$$

Using this product for min and max quantifiers to define a min-max operation with y's range depending on values of x to give:

$$(min \otimes^K max)(q) = \min_{x:X} \max_{y:Yx} q(x, y)$$

where min and max are quantifiers of $\min : (X \to R) \to R)$ and $\max : (x : X) \to (Yx \to R) \to R$ respectively.

Definition 2.5. (K-sequence) Like before with Paths and Quantifier trees, we can define a single quantifier as $\bigotimes^K \phi^t : K(P\text{Path}X^t)$ on quantifier trees by induction. This is known as the K-sequence and defined by

1. The quantifier $\lambda q.q\langle\rangle : K(\text{Path}([]))$ is defined for the empty tree.
2. The quantifier $\phi \otimes^K (\lambda x. \bigotimes^K \phi^f x)$ is defined for the quantifier tree $\phi :: \phi^f$.

2.1 Higher-order games

By using all the types of trees defined, a higher-order game (finite otherwise cannot be proved) can be defined by a tuple (X^t, R, q, ϕ^t) where

1. The first element X^t is a type tree denoting the moves that are viable to the player at a specific point in the game.
2. The next element R is all possible results of the game denoted as a set.
3. q is the outcome function defined as $q : \text{Path}(X^t) \to R$.
4. Finally, the element ϕ^t is a quantifier tree.

With $\bigotimes^K \phi^t)(q) : R$ denoting the optimal outcome for the game described.

3 Conclusion

Combining all the definitions described for dependent type trees, they can be generally applied to finite higher-order games. We model the games of Connect 4 as such, for a higher-order game (X^t, R, q, ϕ^t), Connect 4 is defined by the following definition.

Definition 3.1. (Model of Connect 4)

1. The dependent type tree of depth 42 that demonstrate the available columns the players can play their move. Also describes what columns have pieces (which are coloured by red and blue for the 2 players) within them with a max size of 6 (denoting the board). Depending on the first player's column choice with their colour, the 2nd player chooses from the same 7 columns with the other colour unless the column is filled. This is repeated until a player wins or a draw occurs meaning that there are no columns to pick as the board is full. Which leads to the empty tree.

2. The linearly ordered set $-1 < 0 < 1$ represents R.

3. $q : \text{Path}(X^t) \to R$ denotes the outcome function, that given a sequence of column moves with coloured pieces on the board, then it will return 1 or -1 if a player wins and 0 if it ends with a tie, i.e. there are no more moves.

4. ϕ^t is the quantifier tree which uses the min and max quantifiers depending on which players turn it is. If the red player moves first, and the value 1 means that red has won then the quantifiers tree's root is the max quantifier.

Therefore, by defining the types and functions this way, they enable the possibility to express the game with trees so that they can be used to calculate optimal strategies for the players. Which is explored further in [4] with the next stages and definition of selecting function trees, the optimal strategy calculation and the path to attain this goal. Along with this paper, there is also Agda implementation which I have studied and built upon to generate an example with Connect 4. However due to the possibilities in Connect 4, the process is even slower compared to the Tic-Tac-Toe version so further development is required to ensure that the program is more efficient.

4 Further Works

Another idea similar to this is the exploration of simultaneous games with the use of matrices rather than trees. Other games that could not be defined before can be unlocked through this further implementation. Where the strategies of players can be converted to a payoff matrix. This means that all strategies are known to the players and a singular choice must be made that benefits them best. Consequently

type definitions can be written in relation to matrices to provide an optimal strategy and best choice for each player reaching an equilibrium to the matrix game. Nash equilibrium [5] is a mathematical concept that can be applied to the results generated by matrix definitions which determines if the payoffs of both players have reached optimality. I.e., neither players benefit from changing their strategy. Furthermore, this can be written and proven in Agda [6].

References

[1] Juan C. Burguillo. *Game Theory*, pages 101–135. Springer International Publishing, Cham, 2018.

[2] Martin Escardó and Paulo Oliva. Selection functions, bar recursion and backward induction. *Mathematical structures in computer science*, 20(2):127–168, 2010.

[3] Martin Escardó and Paulo Oliva. What sequential games, the tychonoff theorem and the double-negation shift have in common. In *Proceedings of the third ACM SIGPLAN workshop on Mathematically structured functional programming*, pages 21–32, 2010.

[4] Martín Escardó and Paulo Oliva. Higher-order games with dependent types. *arXiv preprint arXiv:2212.07735*, 2022.

[5] David M Kreps. Nash equilibrium. *Game theory*, pages 167–177, 1989.

[6] Stéphane Le Roux, Érik Martin-Dorel, and Jan-Georg Smaus. Existence of nash equilibria in 2-player simultaneous games and priority games proven in isabelle.

[7] Theodore L Turocy. Texas a&m university. *Bernhard von Stengel, London School of Economics "Game Theory" CDAM Research Report (October 2001)*, 2001.