EXAMENSARBETE INOM TEKNIK,
GRUNDNIVÅ, 15 HP
STOCKHOLM, SVERIGE 2019

Epistemic Structures for Strategic Reasoning in Multi-Player Games

ERIK HANDBERG

LARA ROSTAMI

KTH
SKOLAN FÖR ELEKTROTEKNIK OCH DATAVETENSKAP

Epistemic Structures for Strategic Reasoning in Multi-Player Games

ERIK HANDBERG
LARA ROSTAMI

Bachelor in Computer Science
Date: June 18, 2019
Supervisor: Dilian Gurov
Examiner: Örjan Ekeberg
School of Electrical Engineering and Computer Science
Swedish title: Kunskapsstrukturer för strategiska resonemang i flerspelarspel

Abstract

A game can be visualised with a directed graph, where each node is a game state and the edges are the players' actions that lead to new game states. To obtain the winning condition of the game whilst avoiding the losing condition, the players must perform the correct actions in coalition. The goal is thus to find winning strategies for these games, which ensure that the winning condition is reached. For games with *imperfect* information, the players might not be able to distinguish some game states from each other, which makes finding strategies harder. With the multiplayer knowledge subset construction (MKBSC), one may reduce the original graph to an "expanded" graph, and from this new graph try to find winning strategies, that can later be translated back into the original game graph. In this report, we have investigated how we gain information in each iteration of the MKBSC. This was done by introducing e-trees, a mathematical tree construction that can visualise knowledge of the current game state for each player in the game. It was found that these e-trees can display how the knowledge develops for each player given any iteration of the MKBSC, and that they can provide an intuitive way of understanding how strategies are found in a game.

Sammanfattning

Ett spel kan visualiseras med en riktad graf, där varje nod är ett speltillstånd. Kanterna är spelarnas möjliga drag i spelet, där dragen leder till nya speltillstånd. För att vinna spelet utan att hamna i det förlorande tillståndet måste spelarna utföra de rätta dragen tillsammans. Målet är därför att hitta vinnande strategier i sådana spel, som garanterar att spelarna vinner. För spel med *imperfekt* information kan spelarna inte nödvändigtvis urskilja vissa speltillstånd från varandra, vilket gör det svårare att hitta vinnande strategier. Med den så kallade *multiplayer knowledge subset construction* (MKBSC) kan man reducera grafen för det imperfekta spelet till en enklare graf, och försöka hitta strategier i den nya grafen istället, för att sedan översätta den strategin så att den passar originalgrafen. I den här rapporten har vi undersökt hur man får information när man använder MKBSC, och vad man kan använda den informationen till. Det här gjordes genom att introducera "e-trees", en matematisk trädkonstruktion som kan visualisera kunskapen för varje spelare i spelet. Vi kom fram till att MKBSC kan visualisera hur kunskapen för varje spelare utvecklas vid varje iteration av MKBSC, och att träden ger en mer intuitiv bild över hur strategier kan framställas ur ett spel.

Contents

Chapter 1

Introduction

Certain problems can be simulated with a game, where the problem's solution is the game's winning condition. Consider, for example, a robot performing a task in a factory environment, where the right actions must be executed in the correct order to "win" the game. Each action leads to a *game state*, which describes the state the game is in. We can visualise this game in a *directed graph*, where the robot's actions are edges, and the game states are the nodes.

In a game of chess, each player knows about every state in the game. The player knows that one move will lead to a certain game state. We call this a game of *perfect* information. In a game of poker however, the player might not be able to obtain every piece of information there is in the game. The player might not know what cards are held by the other players, which introduces a different type of strategy as compared to chess. We call poker a game of *imperfect* information. In this report, we will focus on the latter type of games.

The goal of a game is to reach a certain objective, which is possible with a strategy. If we assume the players have memory in a game of imperfect information, finding a strategy becomes a complex problem that could require a lot of computational power to solve. However, if we assume the players have no memory, finding strategies or concluding that there are none becomes a simpler problem.

To help in the process of finding winning strategies, we can use the *Multi-player Knowledge-Based Subset Construction* (MKBSC). The idea of MKBSC is to make use of the knowledge a player has about itself and the other players.

Each iteration of the construction computes each player's knowledge about the possible state of the game and generates a new game based on that knowledge. In a sense, it allows players to consider not only their own knowledge, but also the other players' knowledges. In a sense, the MKBSC creaets a nesting of knowledge, where the players can reason about the other players, and for each iteration we allow the players to consider another level of knowledge. This nesting of knowledge can just after a few iterations become complex and difficult to comprehend.

When the construction has been iterated on a game one or multiple times, it might be possible to find an *epistemic strategy*. Instead of looking at where a player is at in the game, the player now only uses what knowledge it has about itself and possibly about other players to make the decision on its next move. Epistemic strategies are a natural way to represent strategies for players. Imagine for example a poker player that doesn't just look at the cards in its hand and on the table, but also considers what the other players knows before making a move.

A tool (we will refer to this tool as the MKBSC-tool), available at [1], has previously been created that can apply MKBSC on games, and print out the knowledge that players receive for each iteration of the MKBSC. The issue with the MKBSC-tool, however, is that this nesting of knowledge becomes hard to deconstruct and visualise. It is thus the goal of this paper to create a new way of visualising knowledge of the players in a game as a result of having applied MKBSC in the game.

1.1 Research Question

An idea of representing the nesting of knowledge with the MKBSC is to make use of trees. In this paper, we will attempt to visualise the nesting of knowledge with the MKBSC with tree constructions.

1. For each iteration of the MKBSC, the nesting of knowledge about a player's current location becomes larger and more difficult to understand. How can we create a new way of representing this nesting of knowledge?

2. In previous works on the subject [2, 3], a tool [1] has been implemented to allow users to implement and visualise the MKBSC on a game graph. How can we extend this tool, as to make it display our knowledge construction specified in the item above?

3. Given that the items above were succeeded, does our knowledge construction aid us in finding epistemic strategies?

Chapter 2

Background

We begin by defining important concepts central to this paper.

2.1 Games on finite graphs

A finite game can be depicted on a directed graph. A player starts on a starting node, and has, through various actions, the ability to traverse through the graph. The objective of the game depends on how it is defined. We consider the games as if there are players playing against one single entity, which we call “nature”.

2.1.1 Single player games with perfect information

For depicting games on directed graph, we follow the notation in [4].

Game graphs We denote the *game graph* with the tuple

$$G = (L, l_0, \Sigma, \Delta),$$

where:

- L is a finite set of states (the nodes in the game graph)
- $l_0 \in L$ is the initial state of the player
- Σ is the set of actions available to the player
- $\Delta \subseteq L \times \Sigma \times L$ is a set of labelled transitions (the labelled edges in the graph)

An important detail here is that as the player traverses through the graph, the player always knows their own location. Possible transitions for a player, given a set of states $S \subseteq L$ and an action $\sigma \in \Sigma$, are defined as

$$Post^G_\sigma(S) = \{l' \in L : l \in S \text{ and } (l, \sigma, l') \in \Delta\}$$

and will return all accessible states from the set of states S using the action σ.

An example of a situation modelled as a game graph is shown in figure 2.1. This is the same example used by Jacobsson and Nylén [2], but with a different context. In the example, a company making bird houses have automated the part of the factory drilling the entry hole to the houses. A robot has been tasked with drilling entry holes in bird houses. The robot is given either houses with entry holes or houses without entry holes. The robot should only pass on houses with entry holes, meaning that it first has to drill a hole in the houses where it is missing.

Winning condition To succeed in the game, the player must fulfill the winning condition of the game. In this paper, we will only consider *reachability* conditions, meaning each graph has a state which the player must reach in order to win. A graph can also have a losing state, meaning if the player reaches this losing state, the winning state can never be reached [4].

Strategies We denote a play in G as an infinite sequence $\pi = l_0 l_1 \dots$ such that $\forall i \geq 0$, there exists σ_i such that $(l_i, \sigma_i, l_{i+1}) \in \Delta$. A history is a "cut" from a play such that $\pi(i) = l_0 \dots l_i$. We denote the set of histories as L^+ .

We can now define a strategy. A strategy in G for the player is a function $\alpha : L^+ \to \Sigma$ that maps histories to actions. A strategy is *memoryless* if the strategy only depends on the current node the player is in, meaning the player cannot refer to previous moves to make a decision of the next move. If the strategy requires some finite amount of memory it is called a *finite-state strategy* [4].

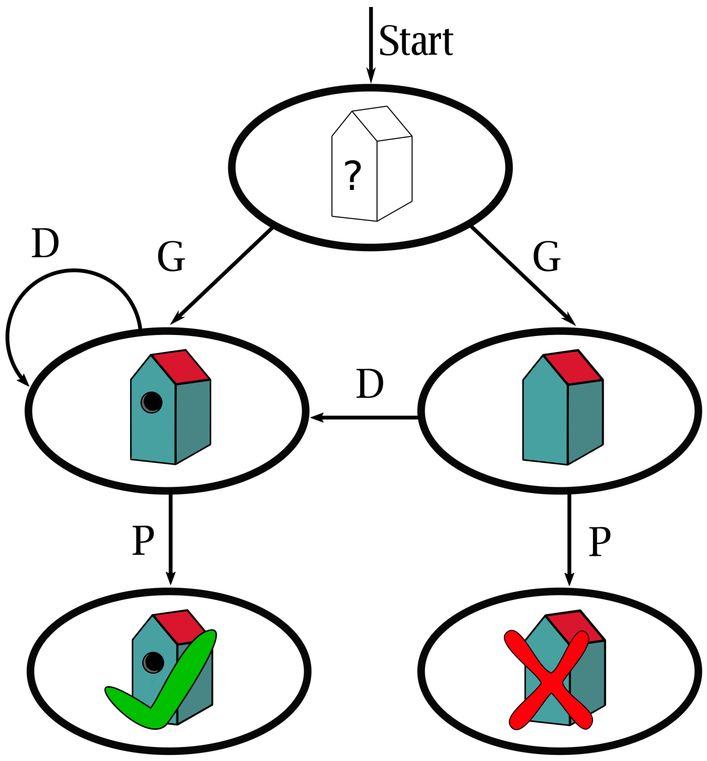

Figure 2.1: A situation with automated drilling of entry holes in bird houses, modelled as a game graph. A robot receives newly assembled houses, some with entry holes and some without entry holes. The robot must drill entry holes in the houses where this is missing. After receiving a house, the robot can either drill a hole (action $\sigma = D$) or pass it on down the construction line (action $\sigma = P$). The winning condition is to pass on houses with drilled entry holes, while passing on a house without an entry hole is the losing condition. The states are $L = \{\texttt{start, hole, no hole, win, lose}\}$ where start is the initial state l_0. The set of actions available to the player (the robot) is $\Sigma = \{G, D, P\} = \{\texttt{Get bird house, Drill entry hole, Pass on bird house}\}$

2.1.2 Single player games with imperfect information

We now consider games of *imperfect* information. In such a game, the player cannot distinguish some locations from each other, meaning the player may not necessarily know which state the game is in. In this case, we partition the game states into *observations* o, and we call the set of observations $\mathcal{O}$. The states in a given observation are indistinguishable from each other to the player. Note that the observations themselves are still distinguishable from each other. We denote this game as

$$G = (L, l_0, \Sigma, \Delta, \mathcal{O})$$

This game is played the same way as in the case with perfect information, however the player must now account for that only observations of the game are visible, not the locations.

Note that with this definition we can still simplify it to a perfect information game. If all observations contain a single location, then all locations are distinguishable from each other, which is the definition of a perfect game [3].

Consider again the example with the robot drilling entry holes in bird houses. Some time after the robot has been installed, a sensor on the robot malfunctions, leaving it unable to detect whether or not an entry hole has already been drilled. This creates a situation that, when is modelled as a game graph, is a game of imperfect information, see figure 2.2.

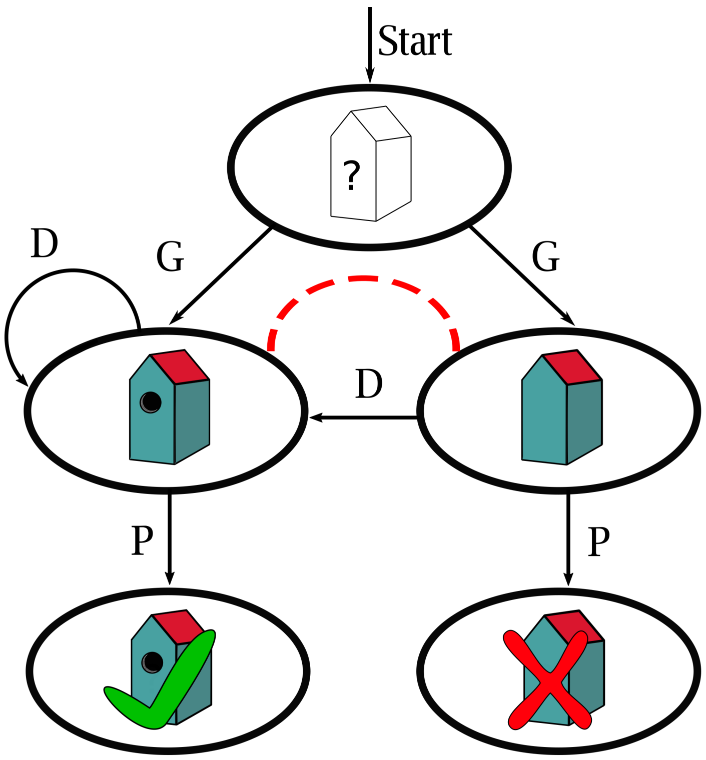

Figure 2.2: The same situation as figure 2.1, but with the modification that the robot now can't tell the difference between when an entry hole already has been drilled or not (marked with the red striped line), making it a game of imperfect information. The set of states L and the set of actions Σ are identical to the situation in figure 2.1, but because this is a game if imperfect information there is now also a set of observations $\mathcal{O} =$ `{{start}, {hole, no hole}, {win}, {lose}}`.

2.1.3 Multiplayer games with imperfect information

A multiplayer game of imperfect information plays similarly to a single player game of imperfect information, and is defined as

$$G = (L, l_0, \Sigma, \Delta, \mathcal{O})$$

with the difference from the single player case that

- $\Sigma = \Sigma_0 \times \Sigma_1 \times \cdots \times \Sigma_{n-1}$
- $\mathcal{O} = \mathcal{O}_0 \times \mathcal{O}_1 \times \cdots \times \mathcal{O}_{n-1}$

where n is the number of players. A player i has thus the actions Σ_i and the observations $\mathcal{O}_i$ available to them. This means that players could have different information and actions available to them in a certain state of the game.

Strategies in a multiplayer game consist of the joint strategies of the players in the game. In the case of a reachability condition, the players usually share the same winning condition. Note that the players may not share any information between each other: we assume no communication in these games.

As an example, again consider the case of the bird house factory. Now, the drilling of entry holes is instead handled by two robots working together. One of the robots has a working sensor while the other robot has a malfunctioning sensor, meaning that only one of them can tell the difference between a bird house with an entry hole and a bird house without an entry hole, see figure 2.3.

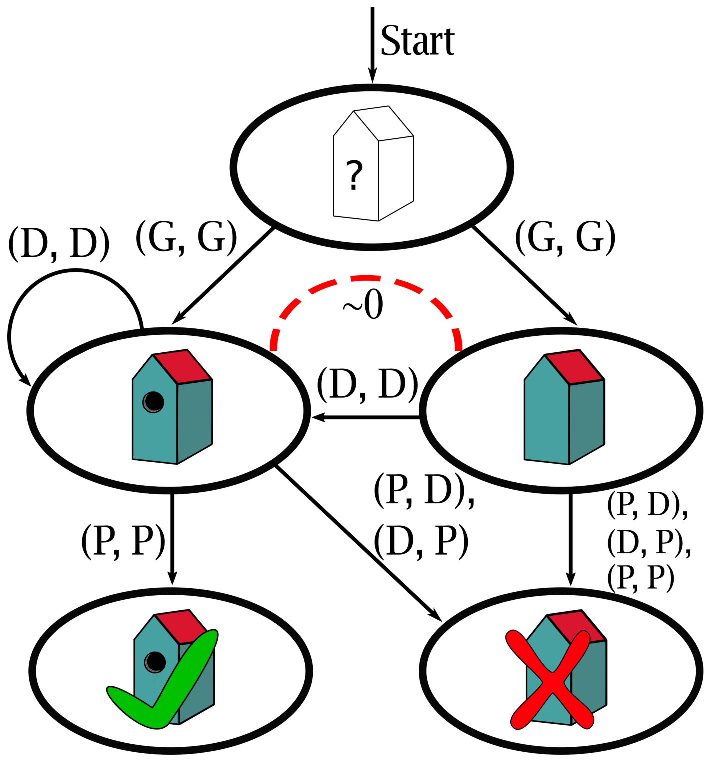

Figure 2.3: The similar situation as in figure 2.1 and 2.2, but with two robots working together. In this version of the game player 0 can't tell when a bird house already has an entry hole drilled or not. Transitions in this game are written as tuples, where the i:th entry in the tuple represents the action of player i.

Projecting games From the definition of a multiplayer game with imperfect information, it should be clear that each player *sees* the game differently, meaning their observations may differ from each other. To see how a player sees the game, we can *project* the game G onto player i, which creates a new game graph, labelled $(G|i)$, from the perspective of player i.

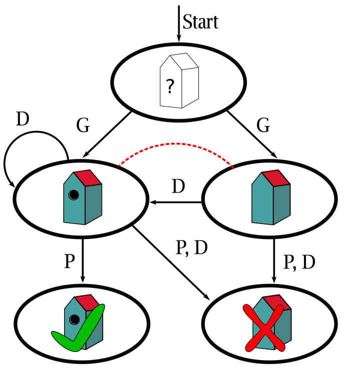

Figure 2.4: The game $(G|0)$ as seen in figure 2.3, projected on player 0. The red line shows that the player cannot distinguish between these two states.

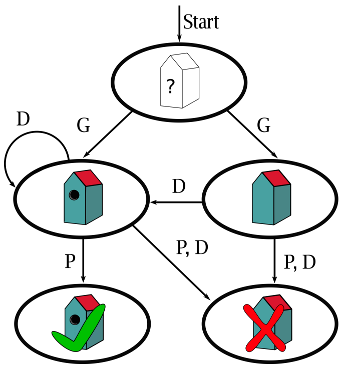

Figure 2.5: The game $(G|1)$ as seen in figure 2.3, projected on player 1. In this case, the player can distinguish all states from each other.

2.2 Representing knowledge in games

We begin this section by recalling that all perfect games can be represented as an imperfect game with all observations being singletons, meaning we need only to consider imperfect games in this section.

Representing knowledge As mentioned in section 2.1, the state a player can be located in is denoted as l_i, where $l_i \in L$. Suppose now that the state l_i is the only state related to a observation, meaning the player can clearly distinguish if the player is located in this state or not. The *knowledge* of the player is represented as the set of the states in the observation, in this case this would be represented as $\{l_i\}$. By following this logic, we can extend it to multiple states. If the states $l_i, l_j, l_k \ldots$ are in the same observation (meaning the states are indistinguishable from each other), we would denote the player's knowledge as $\{l_i, l_j, l_k \ldots\}$.

Epistemic depth In a game, the knowledge a player has about the state of the game, both its own knowledge and and its knowledge of other players' knowledge, can be allowed to stretch deep. One way to describe this, as done by Lundberg [3], is to define different *levels* of epistemic reasoning. At level 0, players do not have any knowledge as of what states of the game it and and other players can be at, but can only perform an action based on its observation. At level 1, players are allowed the knowledge of what states are possible based on the current observation. Not until level 2 are the players allowed to reason about the (level 1) knowledge of other players. Generally, players are allowed to reason about *k* levels of knowledge of other players at level $k+1>0$.

Epistemic trees Since, for levels $k>1$ of epistemic reasoning, a player is allowed to reason about the knowledge of every other player, the possible knowledges can branch out further as the epistemic level increases. An intuitive representation of this is with a tree. One such epistemic tree structure is mentioned in section 2.4.1.

2.3 Knowledge-based algorithms on games

2.3.1 Knowledge based subset construction

The knowledge based subset construction (KBSC) is the idea of transforming the game graph into a graph of knowledge. We convert the game graph into what the the player's knowledge of the game is in each state, and can with this new information try to find a strategy in the game graph that might not have been intuitively visible to us from the original game graph [2].

KBSC We use the definition in [3]. Given a game $G=(L,l_0,\Sigma,\Delta)$ of *perfect* information, the knowledge based subset construction (KBSC) of G is defined as the perfect game that represents the knowledge the player has in the game. We construct the graph

$$G^K=(\mathcal{L},\{l_0\},\Sigma,\Delta^K)$$

where

- $\mathcal{L}=2^L\backslash\{\emptyset\}$ is powerset of L (the set of all subsets of L)
- $\Delta^K=\mathcal{L}\times\Sigma\times\mathcal{L}$

This new game graph represents the knowledge the player has in the game. Note that all locations will have been transformed to *knowledge sets* which we denote as $\{l_i\}$, which represent the knowledge the player has in this game state.

2.3.2 Knowledge based subset construction for multiplayer games

The multiplayer knowledge subset construction, proposed by [3], involves applying KBSC to each player individually, and then combining these graphs to form G^K.

MKBSC We start by projecting the game onto each player, which creates the new graphs $(G|i)$.

We apply KBSC to each individual player graph, yielding $(G|i)^K$. We then combine each $(G|i)^K$ for each player to yield G^K. We can thus define $G^K = (G|0)^K \times (G|1)^K \times \cdots \times (G|n-1)^K$.

Note that this new game graph involves all possible *combination* of game states, but they might not be *reachable* or *consistent* in the context of the game. We thus use a pruning operation ρ on $(G|0)^K \times (G|1)^K \times \cdots \times (G|n-1)^K$ to yield the final game graph, meaning

$$G^K = \rho((G|0)^K \times (G|1)^K \times \cdots \times (G|n-1)^K)$$

Figure 2.6 displays an example of using the MKBSC on game G in figure 2.3, which is done by combining figure 2.4 and 2.5 with each other. Notice that player 1 has gained an observation between the states $\{\{\texttt{no hole, hole}\}, \{\texttt{hole}\}\}$ and $\{\{\texttt{hole}\}, \{\texttt{hole}\}\}$. If player 1 knows the game is in the state $\{\texttt{hole}\}$, it cannot determine whether player 0 thinks the game is in the state $\{\texttt{no hole, hole}\}$ or $\{\texttt{hole}\}$. It should thus be clear that the MKBSC can introduce new observations to the players.

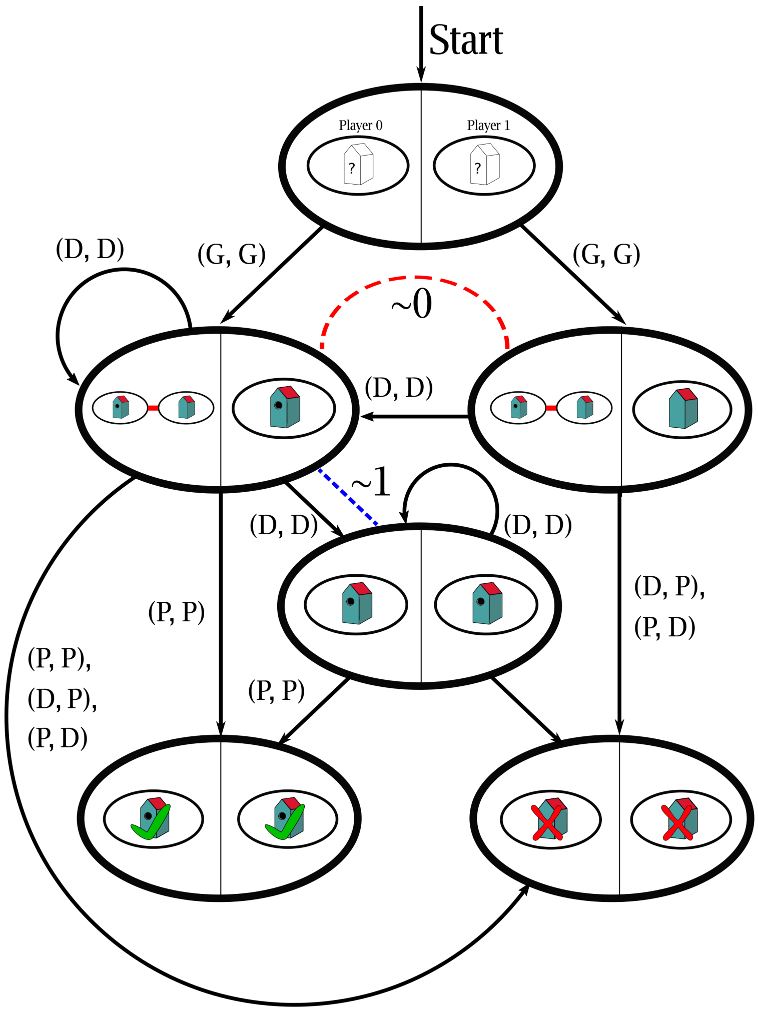

Figure 2.6: The game G^K, where G is the game seen in figure 2.3. Each node represents the composition of knowledge for both players, given that a certain action has been made. Consider, for example, the (G, G) action on the node {`start, start`}. We could either end up in the node {{`hole, no hole`}, {`hole`}} or {{`hole, no hole`}, {`no hole`}}. Notice that the first element of these sets represent what player 0 knows, and the second element represent what player 1 knows.

Notice that these observations take different epistemic meanings depending on how many times the MKBSC has been applied. In the graph G, an observation between two states translates to a player not being able to tell the difference between them. In the graph G^K, where the MKBSC have been applied once, an observation between two states translates to a player not being able to tell the difference between what knowledge the other player possesses in these games. For graphs G^{jK} where the MKBSC has been applied $j > 1$ times, an observation between two states translates to a player not being able to tell the difference between the other players knowledge of the other players knowledge, and so on.

Strategy recovery When trying to find winning strategies in a game using the knowledge gained from the MKBSC, the goal is to look for memory-less strategies in the games resulting from the MKBSC. If such a memory-less strategy can be found it can always be translated back to a finte-memory

2.3.3 A tool for applying MKBSC on games

In 2018 a Python library was developed by August Jacobsson and Helmer Nylén [2], available at [1]. The tool can visualise any game graph and apply the MKBSC to it. Currently, when visualising a game graph, the default option for presenting the knowledge of a player in a certain node is with a string using a specific format designed by Nylén and Jacobsson. This string representation works well for games that only has been iterated with the MKBSC a few times, but as the game is further iterated the string representation of the knowledge of a player becomes increasingly unreadable. We will refer to this tool as the **MKBSC-tool**.

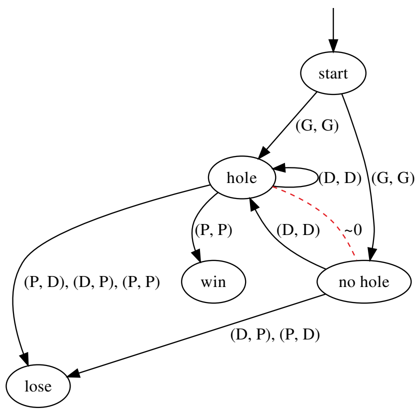

Figure 2.7: The graph G resulting from the implementation of the bird house example in 2.3, drawn with the MKBSC-tool.

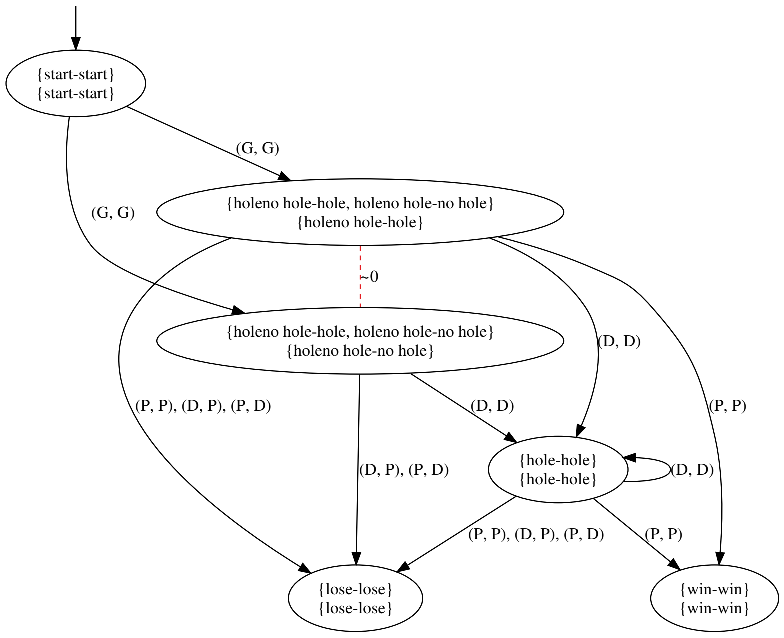

Figure 2.8: The graph G^{2K}, resulting from having applied the MKBSC on G in figure 2.7 two times. After a few iterations of the MKBSC, the string output in each node becomes large and difficult to deconstruct as a reader.

2.4 Related Work

2.4.1 k-trees

One example of a tree representation of knowledge has been defined by Ron Van Der Meyden, called *k-tree* [5]. The structure of these trees is defined using mutual recursion, where a k-tree is a tuple $\langle l, U_1, \ldots, U_n \rangle$. We denote $l \in L$ as a state in the set of states L, n as the number of players and U_i represents the knowledge of the player *i* at level k. In a 0-tree no player has any knowledge of the state of the game, meaning that all U_i in $\langle U_1, \ldots, U_n \rangle$ are all equal to the empty set. In a 1-tree, each U_i represents the knowledge of player i about the state of the game. For k-trees with higher k each U_i represents the knowledge of player i about both the state of the game and the knowledge that other players have about the state of the game [5].

Chapter 3

Method

3.1 Idea

As mentioned before, one issue with the MKBSC-tool is that after a few iterations of the MKBSC, the string representations of a player's knowledge in the nodes in the game graph becomes too large (see figure 2.8). The purpose of this report is to combine the idea of representing knowledge as trees (see section 2.2) with the tool for applying the MKBSC.

We will formally define a new tree structure, based on the work in [5], for representing knowledge of players in games. Furthermore, we will extend upon the current MKBSC-tool as to allow it to represent players' knowledge with such trees. The idea for the tree structure is new and was presented to us by our supervisor.

3.2 Approach

We will introduce a new recursive definition of a tree structure, which one can use to construct the knowledge tree based on the knowledge the player has in a specific game state.

Using this definition, we will then implement this definition in the MKBSC-tool, to allow it to display the knowledge of each player as trees, as an alternative to the current string representations (as seen in 2.8). This will be done by adding a function to the existing python library for translating the knowledge of players in states to a tree representation.

The implementation of the trees in the MKBSC-tool will be done modeling the tree with NetworkX. The image representations of the trees were then created using Graphviz and Imagemagick.

Chapter 4

Results

The results of this paper will be presented as mathematical definitions.

4.1 Knowledge as epistemic states

We will describe the knowledge of the players in a game G^{jK}, where j is the number of times the MKBSC has been applied to the game G and $j \geq 1$. The nodes of the game G^{jK} are called epistemic states, since they are structured objects representing the nested knowledge of the players. We will use E_j as notation for the set of possible epistemic states in the game G^{jK}, but the actual set of epistemic states in G^{jK} will in most cases be much smaller since not all epistemic states in E_j will be reachable (as discussed in section 2.3.2). Each epistemic state $e_j \in E_j$ is a tuple $e_j = \langle S_{0,j}, S_{1,j}, \ldots, S_{n-1,j} \rangle$ where the length n of the tuple is equal to the number of players in G. Each $S_{i,j}$ is a subset of observation in the game $G^{(j-1)K}$. More specifically, $S_{i,j}$ is a subset of an observation for player i at the epistemic depth $j - 1$.

4.1.1 Definition of epistemic states

Definition 1. Let $G = (L, l_0, \Sigma, \Delta, \mathcal{O})$ be an n-player game with imperfect information. The set of epistemic states E_j of depth j are defined inductively as follows:

$$\begin{cases} E_0 = L \\ E_{j+1} = (2^{E_j})^n \end{cases} \tag{4.1}$$

Notice the base case, where E_0 is simply the states in G and $j = 0$ even though we have previously said that $j \geq 1$. In G, no player has yet been allowed to reflect on its knowledge of the game, and therefore the states in E_0 can not be seen as epistemic. Regardless, we choose to use this notation for simplicity.

4.1.2 Deconstructing epistemic states informally

Example 1. Consider the game G^K in figure 2.6, and suppose we have the epistemic state $e_1 = \langle\{\texttt{no hole, hole}\}, \{\texttt{hole}\}\rangle$, see figure 4.1. We would then have

$$e_1 = \langle\underbrace{\{\texttt{no hole, hole}\}}_{S_{0,1}}, \underbrace{\{\texttt{hole}\}}_{S_{1,1}}\rangle$$

Notice that $j = 1$. The epistemic state is therefore at the lowest level of knowledge.

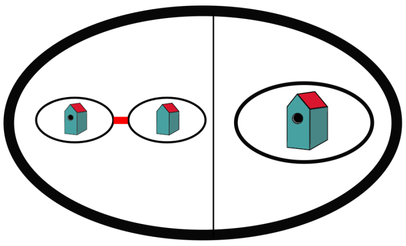

Figure 4.1: A visual representation of the epistemic state e_1 in example 1. This is a node from the multiplayer version of the bird house game where the the MKBSC has been applied once, see figure 2.6.

Example 2. Now consider the game G^{2K} in figure 2.8, and suppose we have the epistemic state $e_2 = \langle\{\langle\{\texttt{h, n}\}, \{\texttt{h}\}\rangle, \langle\{\texttt{h, n}\}, \{\texttt{n}\}\rangle\}, \{\langle\{\texttt{h, n}\}, \{\texttt{h}\}\rangle\}\rangle$, where the node names have been shortened. The components of this epistemic state are deconstructed into

$$e_2 = \langle\underbrace{\{\overbrace{\langle\overbrace{\{\texttt{h, n}\}}^{S_{0,1}}, \overbrace{\{\texttt{h}\}}^{S_{1,1}}\rangle}^{e_1}, \overbrace{\langle\overbrace{\{\texttt{h, n}\}}^{S_{0,1}}, \overbrace{\{\texttt{n}\}}^{S_{1,1}}\rangle}^{e_1}\}}_{S_{0,2}}, \underbrace{\{\overbrace{\langle\overbrace{\{\texttt{h, n}\}}^{S_{0,1}}, \overbrace{\{\texttt{h}\}}^{S_{1,1}}\rangle}^{e_1}\}}_{S_{1,2}}\rangle$$

Notice here that $j = 2$, so we are not at the lowest level of knowledge. We have instead a nesting of knowledge of depth 2.

4.2 Knowledge as e-trees

Similarly to how E_j is used as notation for the set all possible epistemic states in the game G^{jK}, we will use F_j for the set of all possible forests of e-trees in the game G^{jK}. Each forest $f_j \in F_j$ is a tuple $f_j = \langle T_{0,j}, T_{1,j}, \ldots T_{n-1,j} \rangle$ where the length n is equal to the number of players in G. Each $T_{i,j}$ is an e-tree representing the knowledge of player i in that epistemic state.

We will now define an e-*tree*, denoted as $T_{i,j}$, which will represent the elements of an epistemic state e_j in the graph G^{jK}.

Definition 2. An e-*tree* for the ith player is a tree $T_{i,j}$ in the game G^{jK}, and is given by

$$T_{i,j} ::= \text{leaf}(S_{i,1}) \mid \text{tree}(S_{i,1}, T_{0,j-1}, T_{1,j-1}, \ldots T_{n-1,j-1})$$

where

- $S_{i,1}$ is the data kept in the current node.
- n is the number of players
- $T_{0,j-1}, T_{1,j-1}, \ldots T_{n-1,j-1}$ each represent one or more e-trees, where $T_{i,j-1}$ is equal to the empty set.

Intuitively, $T_{i,j-1}$ is equal to the empty set because what a player knows about itself is not represented as a branch in the next, lower level of the tree. Instead, this knowledge is kept in the current node, more specifically in $S_{i,1}$.

The reason why each e-tree $T_{k,j-1}, 0 \geq k \geq (n-1)$

4.2.1 Recursive definition of e-trees

We will now define a recursive method for creating an e-tree from an epistemic state. We begin by defining a helpful function before we define e-trees.

Definition 3. The function node : $S_{i,j} \mapsto S_{i,j}$ extracts the knowledge the player i possesses, and is given by

$$\begin{cases} \text{node}(S_{i,1}) = S_{i,1} \\ \text{node}(S_{i,j}) = \text{node}(S_{i,j-1}) \end{cases} \tag{4.2}$$

where $S_{i,j-1}$ is the ith element from any set $e_{j-1} \in S_{i,j}$.

It might seem odd that we can pick *any* set in $S_{i,j}$ and choose the ith element from it. Intuitively, this is explained by a player i in any epistemic state e_j always having the same knowledge about itself (look for example at the knowledge of player 0 in example 2). Since we for this helper function only need the information that a player has about itself, it does not matter which epistemic state e_{j-1} we choose to look at.

We can now define e-trees.

Definition 4. An e-tree can be generated by the function $\chi_{i,j} : S_{i,j} \mapsto T_{i,j}$, and is given by

$$\begin{cases} \chi_{i,1}(S_{i,1}) = \text{leaf}(S_{i,1}) \\ \chi_{i,j}(S_{i,j}) = \text{tree}(\text{node}(S_{i,j-1}), \{\chi_{k,j-1}(e_{j-1}(k)) \mid 0 \leq k < n, k \neq i, e_{j-1} \in S_{i,j}\}) \end{cases} \tag{4.3}$$

where

- $S_{i,j-1}$ denotes the ith element from the tuple $e_{j-1} \in S_{i,j}$.
- $e_{j-1}(k) = S_{k,j-1}$ is the kth element of the tuple e_{j-1}

We now connect e-trees with forests.

Definition 5. Given an epistemic state $e_j = \langle S_{0,j}, S_{1,j}, \ldots, S_{n-1,j} \rangle$, we can create a tuple of e-trees $f_j = \langle T_{0,j}, T_{1,j}, \ldots T_{n-1,j} \rangle$ that collect the information for all players. We define the function $\chi : e_j \mapsto f_j$ such that

$$f_j = \chi(e_j) = \langle (\chi_{0,j}(S_{0,j}), \chi_{1,j}(S_{1,j}) \ldots \chi_{n-1,j}(S_{n-1,j})) \rangle \tag{4.4}$$

4.2.2 Deconstructing an e-tree informally

The following is an informal way to deconstruct the e-tree.

1. We start at a node, which we call the head. This is the knowledge player i knows about its own location.

2. For each edge connected to the head, we have a label, j. The label represents player j.

3. Through this edge we reach another node, which we call the child. What we can read from this is the knowledge that player i knows about what state player j is in.

4. Repeat from step 1 for child node with player j.

Example 3. Consider the epistemic state

$$e_1 = \langle \underbrace{\{\texttt{hole, no hole}\}}_{S_{0,1}}, \underbrace{\{\texttt{hole}\}}_{S_{1,1}} \rangle$$

in example 1. We shall now retrieve f_1 from this epistemic state, using definition 5.

$$f_1 = \chi(e_1) = \langle \chi_{0,1}(S_{0,1}), \chi_{1,1}(S_{1,1}) \rangle$$

From definition 4, we retrieve

$$\begin{cases} \chi_{0,1}(S_{0,1}) = \text{leaf}(S_{0,1}) \\ \chi_{1,1}(S_{1,1}) = \text{leaf}(S_{1,1}) \end{cases}$$

We thus get the result

$$f_1 = \langle \overbrace{\text{tree}(\text{leaf}(\{\texttt{hole, no hole}\}))}^{T_{0,1}}, \overbrace{\text{tree}(\text{leaf}(\{\texttt{hole}\}))}^{T_{1,1}} \rangle$$

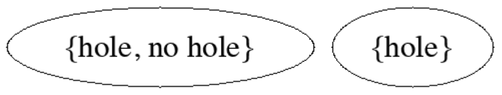

Figure 4.2: The tree representation of $f_1 = \chi(e_1)$, where e_1 is epistemic state from example 1. There are no edges in this graph, so we have no nesting of knowledge.

Example 4. Consider the epistemic state

$$e_2 = \langle \{\langle \{\texttt{h, n}\}, \{\texttt{h}\} \rangle, \langle \{\texttt{h, n}\}, \{\texttt{n}\} \rangle\}, \{\langle \{\texttt{h, n}\}, \{\texttt{h}\} \rangle\} \rangle$$

in example 2. We will now retrieve f_2 from this epistemic state.

$$f_2 = \chi(e_2) = \langle \chi_{0,2}(S_{0,2}), \chi_{1,2}(S_{1,2}) \rangle$$

From definition 4, we retrieve

$$\begin{cases} \chi_{0,2}(S_{0,2}) = \text{tree}(\text{node}(\{\texttt{h, n}\}), \chi_{1,1}(\{\texttt{h}\}), \chi_{1,1}(\{\texttt{n}\})) \\ \chi_{1,2}(S_{1,2}) = \text{tree}((\text{node}\{\texttt{h}\}), \chi_{0,1}(\{\texttt{h, n}\})) \end{cases}$$

Since $\text{node}(S_{i,1}) = S_{i,1}$ and $\chi_{i,1}(S_{i,1}) = \text{leaf}(S_{i,1})$, the equation above becomes

$$\begin{cases} \chi_{0,2}(S_{0,2}) = \text{tree}(\{\texttt{h, n}\}, \text{leaf}(\{\texttt{h}\}), \text{leaf}(\{\texttt{n}\})) \\ \chi_{1,2}(S_{1,2}) = \text{tree}(\{\texttt{h}\}, \text{leaf}(\{\texttt{h, n}\})) \end{cases}$$

We thus get the result

$$f_2 = \langle \overbrace{\text{tree}(\{\texttt{h, n}\}, \text{leaf}(\{\texttt{h}\}), \text{leaf}(\{\texttt{n}\}))}^{T_{0,2}}, \overbrace{\text{tree}(\{\texttt{h}\}, \text{leaf}(\{\texttt{h, n}\}))}^{T_{1,2}} \rangle$$

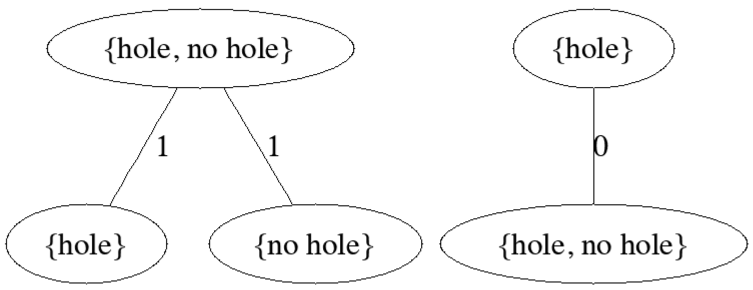

Figure 4.3: The tree representation of $f_2 = \chi(e_2)$, where e_2 is epistemic state from example 4. The leftmost tree represents $T_{0,2}$, and the rightmost is $T_{1,2}$. We use the informal reasoning in the beginning of section 4.2.2 to interpret these trees. Consider $T_{0,2}$. When player 0 knows that the game state is in $\{\texttt{hole, no hole}\}$, it consequently knows that player 1 knows that the game state is in either game state $\{\texttt{hole}\}$ or game state $\{\texttt{no hole}\}$.

4.3 Implementation of e-trees in the MKBSC-tool

The mathematical definitions above were implemented in the existing MKBSC-tool. The difference between the old knowledge structure and the e-tree structure can be seen in the Appendix A. The game used in the example is modeled from the bird house factory situation previously used in the Background. The GitHub repository can be retrieved from [6].

4.4 Using the MKBSC-tool to find strategies

We begin by considering the example in figure 2.7. When the game stars, the bird house is received by both players with the action (G, G). For the next game state, player 1 cannot see whether the bird house already has a hole or not. As seen in the projection $(G|0)$ in figure 2.5, player 0 must consider these two states as the same state. The only way to reach the win state is to pass the bird house, however player 0 cannot distinguish when the bird house is ready to pass. Therefore, there is no memoryless strategy in this game.

We next try to find if there are any strategies in G^K. Applying MKBSC to G with the MKBSC-tool, yields the graph below.

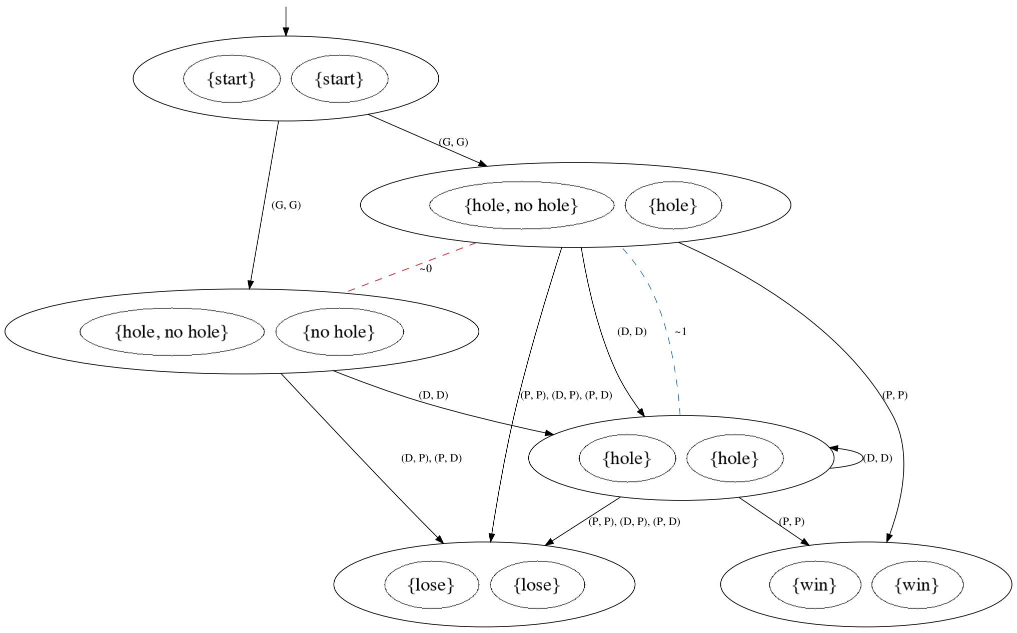

Figure 4.4: The result of applying MKBSC to the game in figure 2.7 with the MKBSC-tool.

Again, the bird house is received by both players with the action (G, G). Consider the case where a bird house with a hole gets passed on. Player 0 cannot distinguish this, however player 1 can. Notice the observation for player 1 for the epistemic states $\langle\{\texttt{hole, no hole}\}, \{\texttt{hole}\}\rangle$ and $\langle\{\texttt{hole}\}, \{\texttt{hole}\}\rangle$. In words, this means player 1 can never be certain if player 0 knows whether the bird house is ready to be passed on or not. Again, this results in the players not knowing when to pass on the bird house, so there is no memoryless strategy in this game either.

Now we check if there are any strategies in G^{2K}. Applying MKBSC to G^K with the MKBSC-tool, yields the graph below.

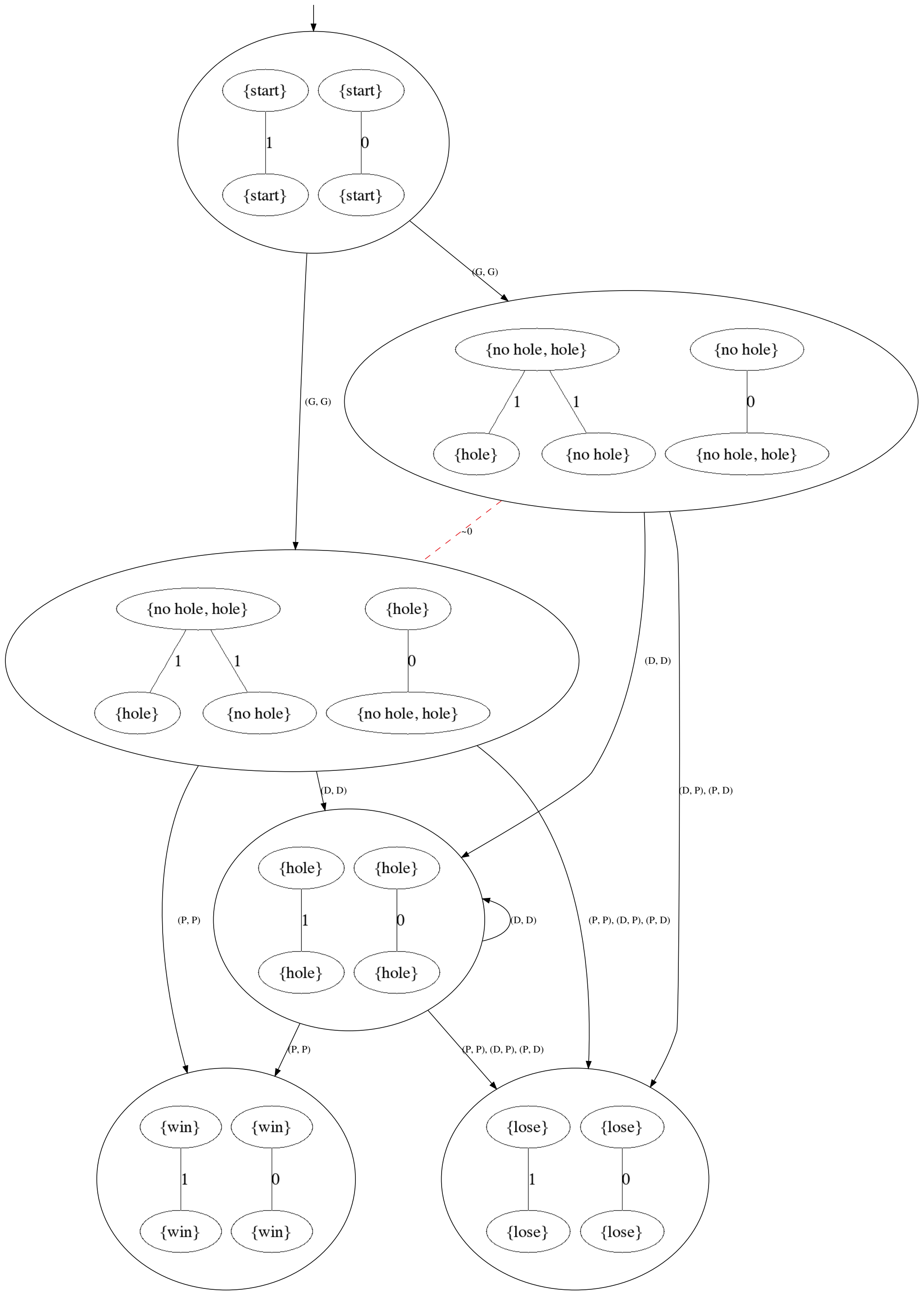

Figure 4.5: The result of applying MKBSC to the game in figure 2.7 with the MKBSC-tool.

We start with the action (G, G). For player 0, the only action, regardless of which state the players end up in, is D. Player 1 can distinguish between the next two states, but again the only action for player 1 is also D, so their joint action after (G, G) would be (D, D). After these actions, the game would end up in the state $\langle\{\langle\{\texttt{hole}\}, \{\texttt{hole}\}\rangle\}, \{\langle\{\texttt{hole}\}, \{\texttt{hole}\}\rangle\}\rangle$. Here, both players know that the game is in game state $\{\texttt{hole}\}$, and that the other player knows about it as well, Thus, from there the players can choose action (P, P) and win the game.

We have thus found a memoryless strategy in G^{2K}. This strategy can then be translated into a finite memory strategy in G, by allowing a memory slot of 2 for each player.

Chapter 5

Discussion

5.1 Improvement

5.1.1 Mathematical definition of e-trees

As seen in section 4, we have managed to successfully implement an abstract definition of e-trees. With these trees, one can visualise the nesting of knowledge much more readily than before.

5.1.2 Implementation of e-trees in the MKBSC-tool

We successfully implemented our definition of e-trees in the existing MKBSC-tool. A visible improvement with the new tool is that the nesting of knowledge in games becomes more structured. Compare, for example, figure A.3 with figure A.4 in the Appendix. Apart from being more readable, we may also see patterns we might not have seen with the old tool. In this case, we can see that some nodes do not gain any new information through the MKBSC, whereas we have two nodes that do gain information through the MKBSC.

5.1.3 Finding strategies for games with the MKBSC-tool

As we saw in section 4.4, the MKBSC along with the e-tree knowledge construction provides an intuitive way of determining whether G^{jK} has a memoryless strategy, and consequently determine if G has a finite memory strategy.

5.2 Limitation

A limitation with our new tool is that it loses information for games where the number of players is greater than 2.

Consider, for example, the epistemic state

$$\begin{aligned} S_2 = \{ & \langle \{ \langle \{\texttt{white}\}, \{\texttt{red, white}\}, \{\texttt{white}\} \rangle \}, \\ & \{ \langle \{\texttt{white}\}, \{\texttt{red, white}\}, \{\texttt{white}\} \rangle, \\ & \langle \{\texttt{red}\}, \{\texttt{red, white}\}, \{\texttt{red}\} \rangle \}, \\ & \{ \langle \{\texttt{white}\}, \{\texttt{red, white}\}, \{\texttt{white}\} \rangle \} \rangle \} \end{aligned}$$

In this example, player 1 is not certain about if player 0 and 2 knows that the game is in {`red`} or in {`white`}. However, player 1 knows that player 0 and 2 knows the same thing, meaning that for example player 1 knows that it is not possible that player 0 knows that the game is in {`red`} while player 2 knows that the game is in {`white`}. Figure 5.1 shows how the corresponding e-tree for that knowledge would look like.

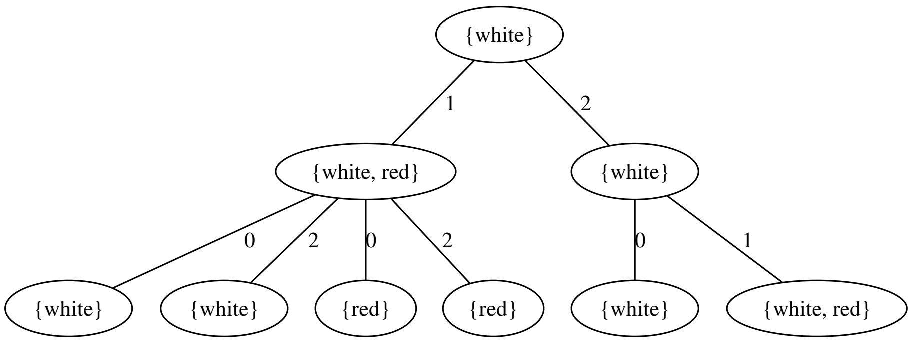

Figure 5.1: An example of an e-tree in a game with three players.

Notice that all four knowledges for player 1 is shown in the tree below the node {`white, red`} in the second level of the tree. However, there is no association between the different knowledges, meaning that there is no way to tell that player 1 can't know that for example player 0 knows {`red`} while player 2 knows {`white`}. That information is lost in this construction of the e-tree in games with more than two players.

5.3 Usefulness of the MKBSC

The usefulness of the MKBSC is uncertain. What we do know is that if a memoryless strategy can be found in a game resulting from the subset construction, it can be translated to a finite-state strategy in the original game. The uncertainty lies in that even though the original game might hold a finite-state strategy, a memoryless strategy might not be found in a game produced by the MKBSC. It is not known which attributes of a game modeled as a graph enables a memoryless strategy to be found as a result of the construction, and therefore there is no way of knowing in advance if the construction will be useful or not.

Modelling each players' knowledge as a tree is an intuitive way of describing how the epistemic depth and each players' knowledge increases for every iteration of the MKBSC. This visualisation could potentially help discovering patterns in game graphs which either guarantees that a finite-state strategy can be found, or patterns that guarantees that such a strategy can't be found. Furthermore, this would in turn help understanding the usefulness of the MKBSC.

Chapter 6

Conclusion

We have investigated whether we can create a new structure for representing the knowledge of each player in a game G when applying MKBSC on G. We conclude that we have done so successfully with our introduction of e-trees.

Furthermore, we have successfully implemented this new construction in the MKBSC-tool.

Finally, we have argued that with our new tree construction, one may be able to more intuitively understand whether or not there is a finite memory strategy in a game G, having found a memoryless strategy in G^{jK}.

6.1 Future work

The authors of this paper would like to see an extension of this tool, to allow it to display games with multiple players better. As discussed in section 5.2, we lose the information regarding how some knowledge states are connected with each other. Our tool could be expanded so that it creates graphs featuring dashed lines or something similar to mark which nodes in the e-tree are connected with each other.

Another extension would be to connect our tool for applying MKBSC to a tool for strategy synthesis. Having done this, one could examine whether there is a clear connection between a game's e-tree and the game's strategy.

Acknowledgements

We would like to thank our supervisor Dilian Gurov, who has guided us in this project.

Bibliography

[1] Helmer Nylén and August Jacobsson. *MKBSC*. `https://github.com/helmernylen/mkbsc`. 2018.

[2] Helmer Nylén and August Jacobsson. *Investigation of a Knowledge-Based Subset Construction for Multi-Player Games of Imperfect Information*. B.S. Thesis. 2018.

[3] Edvin Lundberg. “Collaboration in Multi-agent Games”. MA thesis. KTH, School of Computer Science and Communication (CSC), 2017.

[4] Laurent Doyen, Jean-Francois Raskin, and ENS Cachan. “Games with Imperfect Information: Theory and Algorithms”. In: *Lectures in Game Theory for Computer Scientists* (Jan. 2011), pp. 185–196. DOI: `10.1017/CBO9780511973468.007`.

[5] Ron Van Der Meyden. “Common knowledge and update in finite environments”. In: *Information and Computation* 140.2 (1998), pp. 115–157.

[6] Erik Handberg and Lara Rostami. *MKBSC*. `https://github.com/larasik/mkbsc`. 2019.

Appendix A

Example images from the MKBSC-tool

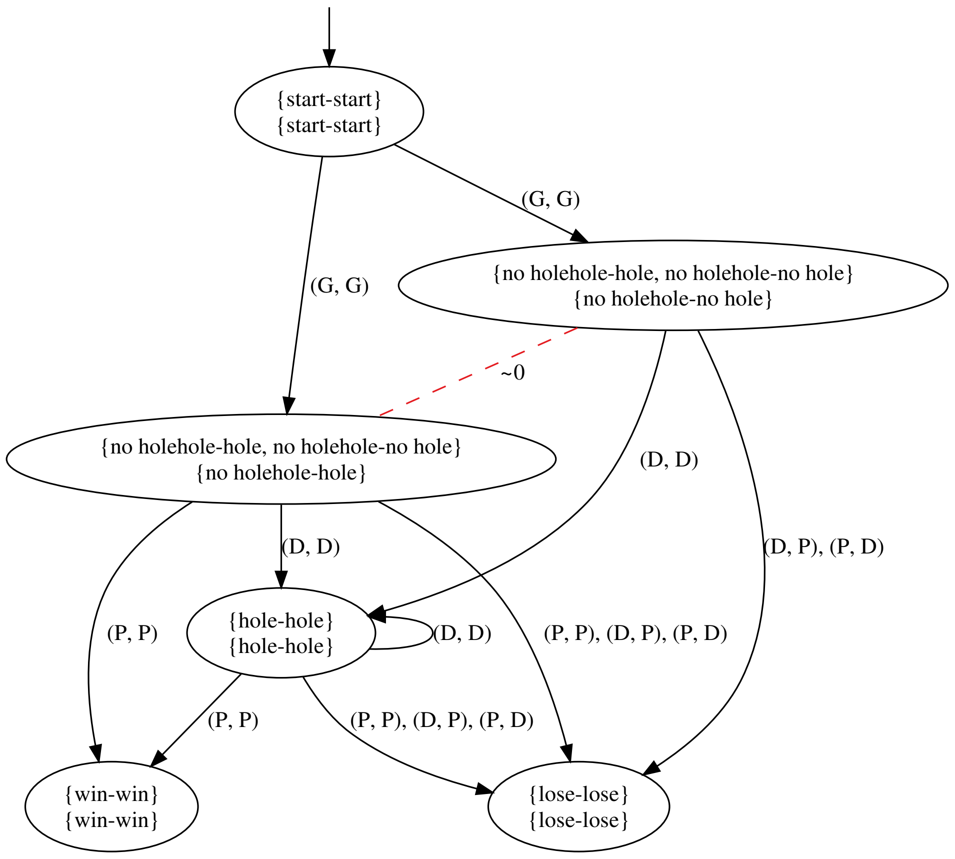

Figure A.1: The graph G^{2K}, where G is the same graph as in figure 2.7. This output is created with the original MKBSC-tool.

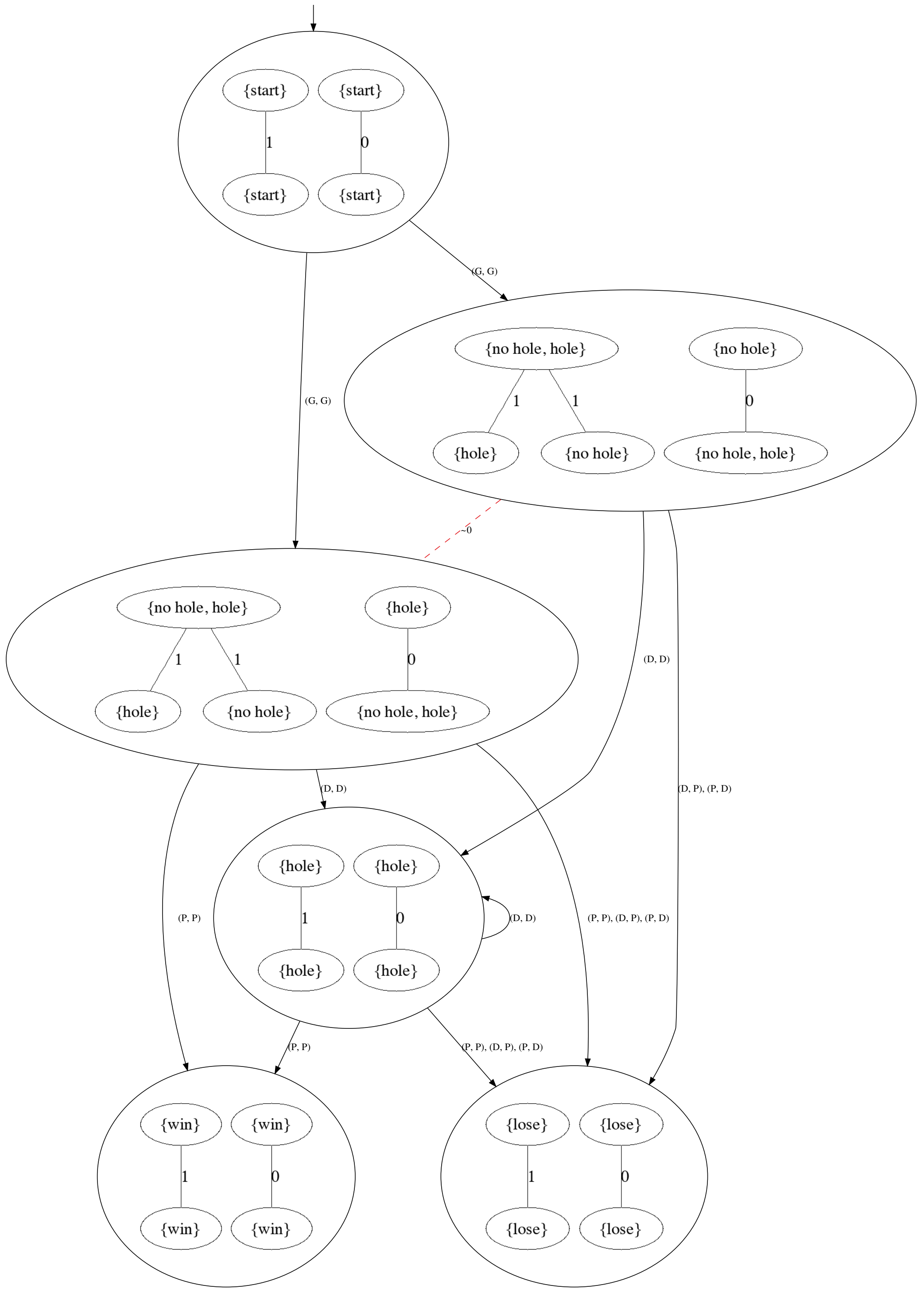

Figure A.2: The graph G^{2K}, where G is the same graph as in figure 2.7. This output is created with the new e-tree formulation within the MKBSC-tool. Compare this output with the output from figure A.1.

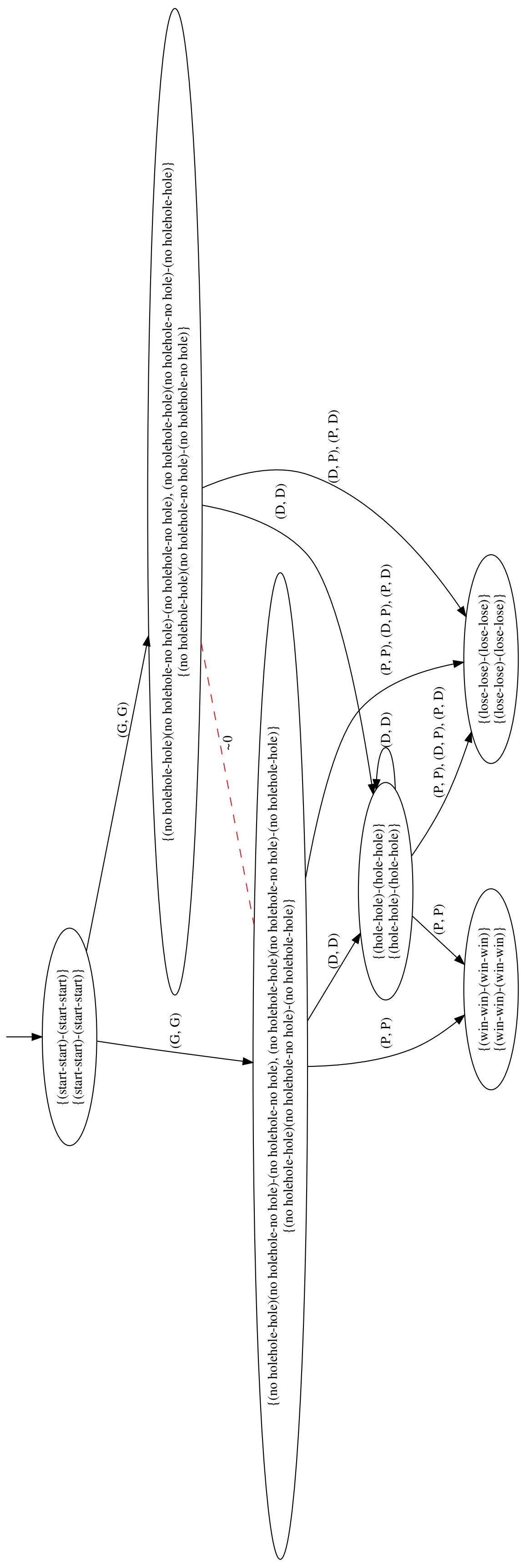
{(start-start)-(start-start)}
{(start-start)-(start-start)}
(G, G)
(G, G)
{(no holehole-hole)(no holehole-no hole)-(no holehole-no hole), (no holehole-hole)(no holehole-hole)(no holehole-no hole)-(no holehole-no hole)-(no holehole-hole)}
{(no holehole-hole)(no holehole-no hole)-(no holehole-no hole)}
{(no holehole-hole)(no holehole-no hole)-(no holehole-no hole), (no holehole-hole)(no holehole-no hole)(no holehole-no hole)-(no holehole-hole)}
{(no holehole-hole)(no holehole-no hole)-(no holehole-hole)}
~0
(D, P), (P, D)
(D, D)
(P, P), (D, P), (P, D)
(D, D)
(D, D)
(P, P)
(P, P), (D, P), (P, D)
(P, P)
{(hole-hole)-(hole-hole)}
{(hole-hole)-(hole-hole)}
{(lose-lose)-(lose-lose)}
{(lose-lose)-(lose-lose)}
{(win-win)-(win-win)}
{(win-win)-(win-win)}

Figure A.3: The graph G^{3K}, where G is the same graph as in figure 2.7. This output is created with the original MKBSC-tool.

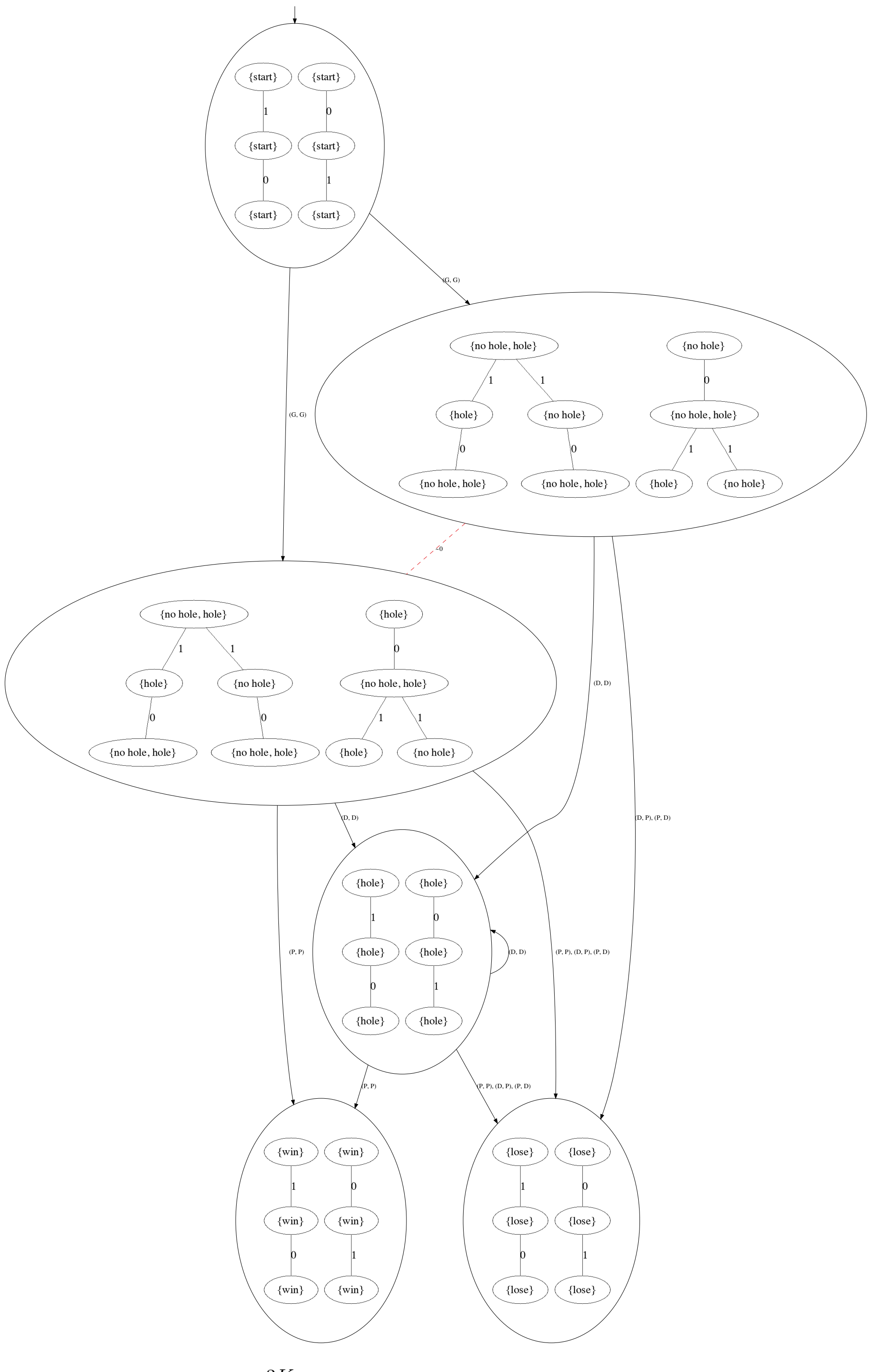

Figure A.4: The graph G^{3K}, where G is the same graph as in figure 2.7. This output is created with the original MKBSC-tool.

TRITA-EECS-EX-2019:325