Automatically translating logical strategy formulas into cognitive models

Jakob Dirk Top (scholar@jakobdirktop.nl)
Institute of Artificial Intelligence, Nijenborgh 9,
Groningen, 9747 AG, the Netherlands

Rineke Verbrugge (l.c.verbrugge@rug.nl)
Institute of Artificial Intelligence, Nijenborgh 9,
Groningen, 9747 AG, the Netherlands

Sujata Ghosh (sujata@isichennai.res.in)
Indian Statistical Institute, 110 Nelson Manickam Road,
Aminjikarai, Chennai 600029, India

Abstract

Whereas game theorists and logicians use formal methods to investigate strategic behaviour, cognitive scientists use cognitive models of the human mind to predict and simulate human behaviour. In this paper, we hope to bring these fields together by creating a translation system which, starting from a strategy represented in formal logic, automatically generates a computational model in the PRIMs cognitive architecture. We run such models to generate response times and decisions made in centipede-like games, a subset of dynamic perfect-information games. Our system is a proof-of-concept for generating cognitive models from formal logic, and presents a new method of otherwise laborious model creation.

Keywords: formal logics, PRIMs, strategic reasoning, automated model generation

Introduction

Centipede-like games

In this paper, we model participants' reasoning in a turn-taking game called Marble Drop (Figure 1). Participants played this game in an experiment against a computer opponent (Ghosh, Heifetz, Verbrugge, & De Weerd, 2017). Because the structure of the game is reminiscent of a centipede (its body extends from top left to bottom right of Figure 1 along the trapdoors and it has five feet corresponding to the bins containing the marbles that are the players' payoffs), such games are dubbed 'centipede-like games'.[1]

Game theory prescribes that players who are commonly known to be rational use the *backward induction* (BI) strategy: one should ignore previous information, and work backwards from the end of the game tree to reach a decision (Perea, 2012). For example, in the 'orange' player's last turn in Game 1 (Fig. 1), he has to decide between going to the left or to the right, for payoffs of 4 or 3 orange marbles, respectively. Using BI, because 4 is more than 3, he chooses to go left, delivering the outcome pair (1,4): 1 for the blue player, 4 for the orange player. One can then continue backwards to compare the left and right choices in the blue player's second turn: going right gives (1,4) while going left gives (3,1); because 3 is more than 1, the blue player would choose to open the left blue trapdoor. One then continues to reason backwards to compare the actions in the orange player's first turn, where the outcome is (1,2) when playing left and (3,1) by playing right. One assumes that, 2 being more than 1, the orange player chooses to open the left orange trapdoor. Finally, one compares the actions in the blue player's first turn, where going left leads to (4,1) and going right leads to (1, 2). Because 4 is more than 1, the blue player will choose to open the left trapdoor to obtain 4 points. Note that playing rationally by backward induction does not necessarily lead to the outcome with the highest sum of players' payoffs – that would have been achieved by both players choosing to open their right trapdoors at all four decision points and ending up with a combined payoff of 6+3.

[1]The games in this paper do not comply with the conditions on payoffs of Rosenthal's original centipede game (Rosenthal, 1981).

Cognitive models

We can investigate human behaviour in centipede-like games by constructing computational cognitive models of the mind and comparing their behaviour to human behaviour. We assume full understanding of the game's rules for both the human players and cognitive models, and investigate their gameplay and the strategies they may be using. We create these models in the PRIMs cognitive architecture (Taatgen, 2013). We recall a few key aspects of PRIMs models. Models in PRIMs have a working memory, used as a mental scratch-pad, and a declarative memory, used for long-term storage of information. Visual information is presented to a PRIMs model hierarchically; the model uses focus actions to move its visual attention through layers of the hierarchy. Models in PRIMs operate by sequentially firing *primitive elements*, which move or compare information present in the model or in the visual input it receives. The process of *production compilation* compiles primitive elements that are often fired in the same sequence into a single production, causing a speed-up when performing the same task multiple times.

Formal logic

In (Ghosh & Verbrugge, online first), a formal logic is proposed which formalizes strategic behaviour as demonstrated by human participants in centipede-like games. A formula describing a strategy, a *strategy formula*, consists of a set of

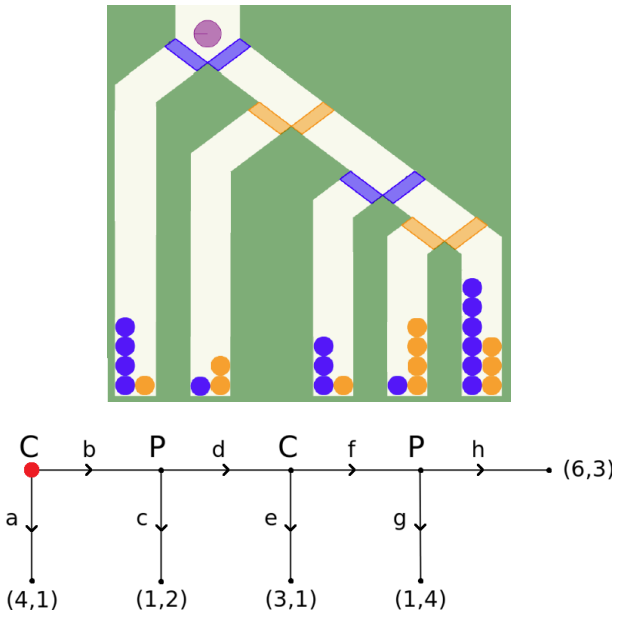

Figure 1: Top: Marble Drop game. Players (assigned blue and orange) control the marble's course by opening the left or right trapdoor of their color once the purple marble arrives there. When it ends up in a certain bin, each player earns the marbles of their color. This example payoff structure corresponds to Game 1 of (Ghosh et al., 2017), see bottom figure. In the payoff pairs, the left payoff is C's and the right is P's.

conditions and an action. If the conditions hold, the action should be played. We use a *myopic strategy* as an example: a player using the myopic strategy only looks at his own payoffs at the current and next ending location. Consider a play of Game 1 starting at player C's first turn. Using the myopic strategy, player C looks at his own payoff should he play a, which is 4, and compares it to his own payoff should he play b and should player P play c, which is 1. The former is larger so player C should play a using the myopic strategy. This case is captured by strategy formula $\mathcal{K}_C^1$ as follows:
$[(\langle a^+\rangle(u_C = 4) \wedge \langle b^+\rangle\langle c^+\rangle(u_C = 1) \wedge (1 \leqslant 4) \wedge \textbf{root}) \mapsto a]^C$

Here, $\mathcal{K}_C^1$ is the name of the formula, with the game number in superscript and the player in subscript. The formula itself is followed by its corresponding player in superscript. The formula consists of conditions, separated by conjunction symbols ($\wedge$), as well as an action, in this case a. The conditions and action are separated by a mapping arrow ($\mapsto$). The first condition, $\langle a^+\rangle(u_C = 4)$, specifies that after edge a is traversed ($\langle a^+\rangle$) from the currently active node (in this case the first node, marked red in Figure 1), player C's payoff should be 4 ($u_C = 4$). The second condition, $\langle b^+\rangle\langle c^+\rangle(u_C = 1)$, specifies that after edges b and c have been traversed, player C's payoff should be 1. The third condition, $(1 \leqslant 4)$, indicates a comparison between these two payoffs. The last condition, **root**, specifies that the currently active node should be the root of the game tree, which ensures that moving across an edge using an operator such as $\langle a^+\rangle$ is possible at this location. If all of these conditions hold, then player C plays a.

Research goals

In this paper we propose a system which creates a PRIMs model from a strategy in the formal logic we just described, capable of playing centipede-like games. Our encompassing goal is to help understand human behaviour in dynamic perfect-information games. Our place in this continuing body of research can be found in Figure 2. Here, human behaviour is found at the top of the diagram, as all research involved aims to understand human behaviour. By observing human behaviour, game theorists formalize strategies as possibly used by human participants. These formal strategies can be used by cognitive scientists to manually construct cognitive models. These models automatically generate data, such as response times. The blue arrow signifies the classic approach of creating cognitive models by hand based on observed human behaviour. The behaviour of such a model can be verified by constructing a behavioural experiment, which gives us data about human behaviour, closing the circle. In the diagram, dashed lines are automated processes. The red dashed line indicates our current research, which automates the creation of cognitive models from formal strategies.

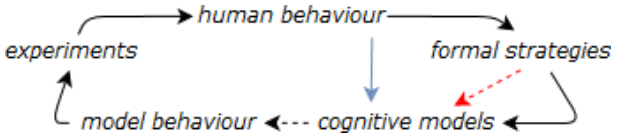

Figure 2: The red arrow in this diagram indicates our place in research, as we aim to automate the creation of cognitive models from formal strategies.

The primary goal of this research is to create a system which automatically generates a fully functioning model in the PRIMs cognitive architecture, capable of playing centipede-like games, from a strategy represented in the formal logic from (Ghosh & Verbrugge, online first). To achieve this goal, we solve several subgoals: first, we create a model in the PRIMs cognitive architecture, capable of playing these games, by hand. We need this model to verify our translation system. Next, we discuss the differences between the formal logic and cognitive models, which is essential to translate one of them into the other. In the next section we describe our solutions and our translation system.

Methods

Verification model

First, we create a cognitive model in the PRIMs cognitive architecture by hand, which uses the myopic strategy as discussed in the introduction. This model plays as player P. We refer to it as the *myopic model*. We only look at the model's first move, in games that either start with player P, or in games where player C has already played action b. We use Game

1 in Figure 1 to demonstrate how this model operates. A model's visual attention always starts at the root of the tree, which is the red dot in the image. The myopic model first moves its visual attention to the first ending location that may be reached. In Game 1, it moves its visual attention across edges b and c. The model then stores its own payoff at this location, the value 2, in its first slot of working memory. The model proceeds to move its visual attention across edges c, d, and e to look at the next ending location. Now the model stores its own payoff at this second location, the value 1, in its second slot of working memory. Lastly, the model retrieves a chunk from declarative memory to compare the values 2 and 1 which it has stored in working memory. The model succeeds in retrieving a chunk specifying that 2 is larger than 1 and, based on this information, ends the game by playing c. This myopic model is generic - it can use the myopic strategy in any centipede-like game. According to the own-payoff strategy, a player should only look at their own payoffs in a game tree, and try to move towards that payoff. For example, in Game 1 in Figure 1, player C should play b and f in an attempt to reach the payoff of 6 at the rightmost node, which is the game's highest payoff. The *own-payoff model* behaves in a manner similar to the myopic model. However, it will also move its visual attention to and make comparisons between its own payoffs further along the game tree. If it discovers a payoff higher than its first payoff, it will move right. If it has compared all payoffs to the first one, and the first one is the largest payoff, it will move down.

Translation system

As mentioned, our new translation system automatically generates a model in the PRIMs cognitive architecture from a formal strategy and its corresponding game. To do so, we first need to represent centipede-like games and formal strategies in our system. For the games, we use the same tree-like structure as seen in Figure 1. Centipede-like games consist of nodes and edges. Nodes can be leaf nodes (ending locations) and non-leaf nodes (player turns). For leaf nodes, both players' payoffs are specified. For non-leaf nodes, the player who has a turn at the node is specified. Edges specify the two nodes they connect. All finite centipede-like games can be stored in this manner. Formal strategies consist of a player, an action, and a list of conditions. Each of these conditions consists of a list of zero or more operators (such as $\langle a^+ \rangle$), and a proposition (such as $(u_C = 4)$). We have already seen most of these propositions and operators in the formula $\mathcal{K}_C^1$.

We now give truth definitions of the relevant logic formulas used in our translation system. The truth of a formula ψ at a node s is defined inductively as follows (see also Ghosh and Verbrugge (online first)). Here, $M = (T, \{\longrightarrow_i^x\}, V)$ is a model consisting of a game tree T, a binary relation on the nodes of the tree corresponding to each node x and each player i (denoted by $\{\longrightarrow_i^x\}$), and a map V assigning to a state s the set $V(s)$ of all true propositions in s.

1. $M, s \models p$ iff $p \in V(s)$ for atomic formulas p.
2. $M, s \models \langle a^+ \rangle \psi$ iff there exists an s' such that $s \stackrel{a}{\Rightarrow} s'$ and $M, s' \models \psi$.
3. $M, s \models \langle a^- \rangle \psi$ iff there exists an s' such that $s' \stackrel{a}{\Rightarrow} s$ and $M, s' \models \psi$.
4. $M, s \models \mathbb{B}_{\mathbb{h}}^{(i,x)} \psi$ iff the underlying game tree T_M is the same as the one for $\mathbb{h}$ and for all s' such that $s \longrightarrow_i^x s', M, s' \models \psi$.

In layman's terms: formulas in the logic are interpreted at the currently active decision node, or the current turn, except when they are preceded by an operator $\langle a^+ \rangle$. Such an operator indicates that the remainder of the statement should be interpreted at the location obtained by following edge a. The proposition **root** is true if the node it refers to is the root of the tree, or the first turn of the game. The proposition $(u_i = q_i)$ is true if player i's payoff is equal to q_i at the node it refers to. The proposition $(r \leqslant q)$ is not interpreted at a specific position, and is true if r is equal to or smaller than q. The proposition $\mathbf{turn}_i$, not found in the formula $\mathcal{K}_C^1$, is true if it is player i's turn at the node it refers to. Finally, we have formulas such as $\mathbb{B}_{g1}^{n1,C} \langle b^+ \rangle c$. This one means 'player C, in Game 1, at the first node, believes that after playing b, c will be played.' It consists of a belief operator $\mathbb{B}_{g1}^{n1,C}$, which accounts for the phrase 'player C, in Game 1, at the first node, believes that'. The operator $\langle b^+ \rangle$ accounts for 'after playing b', and c accounts for 'c will be played'.

Individual components In the present paragraph, we give for each novel component in the logic of Ghosh and Verbrugge (online first) the corresponding behaviour of a PRIMs model generated using this component.

$\langle a^+ \rangle$ **and** $\langle a^- \rangle$: Operators such as $\langle a^+ \rangle$ and $\langle a^- \rangle$ indicate that a proposition should be evaluated at a location different from the current location, and specify which location. A model translated from a formula containing these operators uses focus actions to move its visual attention to the specified location. Focus actions take time to complete similar to human gazing, causing these operators to increase the model's reaction time.

root: When a strategy formula contains the proposition **root**, the PRIMs model will visually inspect the specified node to determine whether it is the root of the tree.

$\mathbf{turn}_i$: When a strategy formula contains a proposition $\mathbf{turn}_i$, where i is C or P, the PRIMs model will read the player name from the specified node in the game tree, and compare it to i.

$(\mathbf{u_i} = \mathbf{q_i})$: The proposition $(u_i = q_i)$ states that player i's payoff is equal to q_i at a certain location. The PRIMs model will compare q_i to a value in its visual input. Because this value may be required for future comparisons, it is also stored in an empty slot of working memory.

$(\mathbf{r} \leqslant \mathbf{q})$: A proposition $(r \leqslant q)$ is a comparison between two values in the game tree. A PRIMs model cannot instantly access each value in a visual display: it has to remember them by placing them in working or declarative memory before it can compare them. A proposition $(u_i = q_i)$ causes

such a value to be stored in working memory. A proposition $(r \leqslant q)$ then sends two of these values from working memory to declarative memory, to try and remember which one is bigger. When a model is created, its declarative memory is filled with facts about single-digit comparisons, such as $(0 \leqslant 3)$ and $(2 \leqslant 2)$.

$\mathbb{B}_{\mathbb{h}}^{(i,x)}$ **and** a: The operator $\mathbb{B}_{\mathbb{h}}^{(i,x)}$ and proposition a are used to construct beliefs about the opponent's strategy. In a belief operator, i is one of the players C or P. The symbol x is the decision node, or turn, where the belief is held. For the four turns in Game 1 (Figure 1) we use *n1*, *n2*, *n3* and *n4*, respectively. Lastly, $\mathbb{h}$ is the game the belief applies to. In Game 1, $\mathbb{h}$ is *g1*. The symbols a through h refer to the actions that can be played in the games, represented as propositions. An example belief is expressed in the following formula:

$$\mathbb{B}_{g1}^{(C,n1)}\langle b^+\rangle\langle d^+\rangle e \tag{1}$$

This formula means 'In Game 1, at node 1, player C believes that after playing of b and d, e will be played'. To verify such a belief, a model employs a strategy similar to the ones used by models in (Stevens et al., 2018). When a model is created, it contains several player-specific strategies in its declarative memory. When a model verifies a belief, it sends a *partial* sequence of actions to declarative memory, corresponding to the assumptions of the belief, in an attempt to retrieve a *full* sequence of actions, which is a strategy. Using Equation 1 as an example, the assumptions of the belief are that b is played. Therefore the model sends the sequence b to declarative memory. The sequences *b-e* and *b-f* could be retrieved, depending on the strategies present in declarative memory. However, only *b-e* verifies $B_{g1}^{(C,n1)}\langle b^+\rangle\langle d^+\rangle e$. All others falsify it.

Problems We encountered two problems in the formal logic in the previous section. First, a comparison $(r \leqslant q)$ does not specify *which* payoffs are being compared, only which two natural numbers (including zero) are being compared. A game containing identical payoffs at different nodes poses a problem for a translation system. Although humans can intuitively determine which comparison would 'make sense', a translation system cannot. Because of this, we use a modified version of the formal logic where each payoff is marked, and each comparison refers to two specific payoffs. This allows a translation system to know precisely which two payoffs it has to pull from working memory to perform a comparison.

Secondly, a belief such as $\mathbb{B}_{g1}^{(C,n1)}\langle b^+\rangle c$ only specifies *what* should be believed, not *how* this belief should be obtained. We use a method similar to (Stevens et al., 2018), where the model begins with strategy chunks in its declarative memory, and verifies a belief by comparing the current game state to these strategy chunks. We have strategy chunks for three strategies: (i) the extensive-form rationalizable (EFR) strategy, which we use because the actions it prescribes correspond more closely than backward induction to the human data of Ghosh, Heifetz, and Verbrugge (2015), (ii) the backward induction (BI) strategy, which we use because it reaches a Nash equilibrium and is often put forward as the game-theoretical solution to games similar to our own (see explanation in Introduction), and (iii) the own-payoff strategy, which is used in previous research on centipede-like games, such as (Ghosh et al., 2017).

According to the EFR strategy, one should consider previous information (Perea, 2012). For example, if player C plays b in Game 1, then player P could rationalize this decision by believing that C has skipped the 4 points obtained by playing a, because C believes that he can get the 6 points at the far right (the only payoff higher than 4). To get these 6 points, player C has to play f as well, which P can use in his decision to play c or d.

Let us explain the model behaviour for a complete strategy formula, namely $\mathcal{K}_C^1$. This formula, as well as the formulas used in previous research using this particular logic, takes the form of a Horn clause, such as $a \wedge b \wedge c \wedge d \rightarrow p$. Given a strategy formula, a PRIMs model tries to verify each proposition in the conjunction sequentially, using the behaviour earlier described for each proposition. If it encounters a proposition it cannot verify, it 'jumps out of' this verification process and *doesn't* play the action prescribed by the formula (what it does we describe in the next section). If the model has verified all the propositions in the conjunction, it plays the action prescribed by the strategy formula.

One problem remains. Conjunctions in formal logic are unordered. Models in PRIMs, however, solve problems sequentially. Therefore we need to order the conjunctions in formal logic, so the corresponding PRIMs model has an order to verify them in. Fortunately, each proposition has to be verified at a specific location. For example, $\langle a^+\rangle(u_C = 4)$, in Game 1 found in Figure 1, has to be verified at the ending location reached by playing a. Eye-tracking data from Meijering, Van Rijn, Taatgen, and Verbrugge (2012) tells us that human participants tend to look through a game tree by following the edges along the shortest path. Therefore we compute the shortest path through the game tree. Our PRIMs models verify propositions as they occur along this path.

Exhaustive strategy formulas In the previous sections we described how a PRIMs model, generated by our translation system, behaves based on a strategy formula. One strategy formula is not always sufficient to describe a strategy. Strategies such as BI and EFR can have multiple solutions if there are payoff ties: in Game 4 in Figure 3, the last two payoffs for P, obtained by playing g and h, are tied, allowing for two options when performing the BI procedure. When using EFR in Game 4, player C playing b instead of a can be interpreted as C going for any of two payoffs higher than the 2 he skipped by playing b. Thus, one has to list all solutions for the specified strategy. Therefore, our translation system allows for a strategy to consist of a list of strategy formulas. The PRIMs model generated from this list tries to verify each formula in it, using the behaviour described in the previous sections, until it finds one it can verify, and play the action prescribed by the formula it verified. There is no need to specify what the

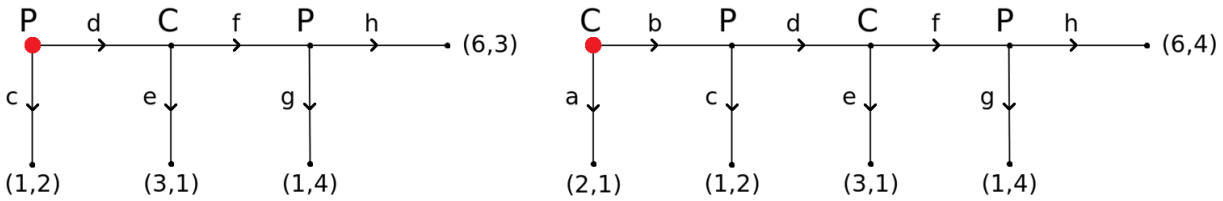

Figure 3: Games $1'$ and 4 from (Ghosh et al., 2017). Game $1'$ is on the left, Game 4 is on the right.

model has to do when it cannot verify any of the formulas: the list is exhaustive, so at least one of them holds.[2]

Experiments

Verification experiment In our verification experiment we compare our handmade with our automatically generated models, based on six different games, three of which can be found in Figure 1 and Figure 3. We have four models: handmade myopic and own-payoff models, and automatically generated myopic and own-payoff models, which are generated from the myopic and own-payoff strategy formulas for Game $1'$. The formula for the myopic strategy is as follows:

$My_P^{1'} : [(\langle c^+ \rangle (u_P = 2) \wedge \langle d^+ \rangle \langle e^+ \rangle (u_P = 1) \wedge (1 \leqslant 2) \wedge \mathbf{root}) \mapsto c]^P$

These models only play Game $1'$ (Figure 3), in the role of player P against computer opponents C playing prespecified moves. We use the same methods as the models in (Ghosh & Verbrugge, online first): we run each model 50 times, where it plays 50 games, to simulate 50 virtual participants who play 50 games each. We record reaction times and decisions.

Differences In our handmade models we have implemented the procedural knowledge required to play the myopic and own-payoff strategies, as described at the beginning of the 'Methods' section. Our automatically generated models, in contrast, play by sequentially verifying propositions in a list and play a certain action if all of them hold, according to a logical formula. Our handmade models are general for any centipede-like game, whereas our automatically generated models are specific for one game. Due to their generality, our handmade models have to look at more properties of the game tree, because no assumptions can be made. For example, for the myopic strategy, when reading the first payoff value from the game tree, for the handmade model, its current goal is 'finding a value in the game tree', its visual attention has to be directed at a leaf node, two slots of working memory have to be empty, one to store the first value, and another has to remain empty for a next value. For the automatically generated model, the game is assumed to be known, as well as the sequence of actions the model has to fire. Therefore, when reading the first payoff value from the game tree, for the automatically generated model, its current goal is 'finding the first value in the game tree', its working memory states that this is the second action performed in this goal, and within its visual input the payoff value the model is currently looking at should be 'two' (in order to verify $\langle c^+ \rangle (u_P = 2)$).

In short, our handmade models are implementations of a strategy, whereas our automatically generated models perform a sequence of actions corresponding to a logical formula, based on this strategy. We compare them to verify our translation system and to investigate the differences.[3]

Exploratory experiment In our second experiment we explore the abilities of our translation system by looking at novel automatically generated models. These models play two games: Game 1 (see Figure 1), and Game 4 (see Figure 3). For both games, we automatically generate two models: one that uses backward induction, and one that uses extensive-form rationalizability. These four models are generated from the BI and EFR strategy formulas for Game 1 and Game 4. These strategy formulas not only contain payoffs and comparisons, like the myopic and own-payoff strategy formulas, but also contain beliefs. We use these games because both the BI and EFR solutions differ between these games, which should give different results. Because some of these strategies have multiple solutions in some of these games, exhaustive strategy formulas are required to describe these strategies, whereas we did not use exhaustive strategy formulas in the previous experiment.

Results

Verification experiment In terms of behaviour, myopic models always play down, and own-payoff models always play right. This corresponds to the strategies these models were generated from. The reaction times for our four models can be found in Figure 4. It indicates that the myopic models are faster than the own-payoff models, and the generated models are faster than the handmade models. Comparing the handmade and automatically generated models, the proportional difference in mean reaction times between the myopic and own-payoff models is similar. The proportion myopic/own-payoff is 0.56 for the handmade models, and 0.55 for the generated models.

[2] The strategy specification language described in (Ghosh & Verbrugge, online first) provides two possible ways of combining strategy specifications, namely, $n_1 + n_2$ (n_1 or n_2) and $n_1 \cdot n_2$ (n_1 and n_2). While the former corresponds to the exhaustive list of strategies, the latter operator is yet to be modelled.

[3] We did not create, by hand, myopic and own-payoff models that play by verifying a strategy formula, for two reasons: firstly, our translation system never fails at generating a cognitive model, and secondly, given a strategy formula, the behaviour of a generated model is fully known to us - writing the same model by hand would be akin to 'copying' the output of our translation system.

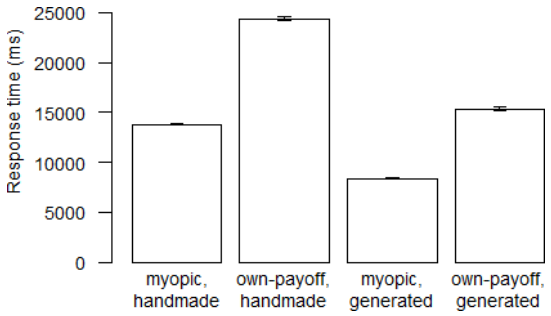

Figure 4: Reaction times of handmade and system-generated myopic and own-payoff models at first decision in Game 1′.

Exploratory experiment The reaction times for our BI and EFR models can be found in Figure 5. Our exhaustive mod-

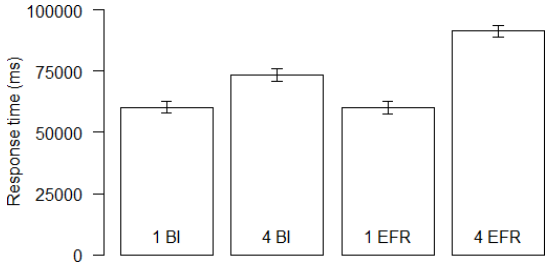

Figure 5: Reaction times for our automatically generated BI and EFR models, when making their first decision. Here, '1 BI' is the reaction time for the BI model in Game 1, etc.

els are a lot slower than our myopic and own-payoff models. Furthermore, reaction times in Game 1 are faster than reaction times in Game 4. It seems that reaction times are a function of the number of formulas required to create the exhaustive strategy formula: for both BI and EFR, in Game 1, only one formula is needed. In Game 4, BI requires two formulas, and EFR requires four formulas. To test this, we perform a simple linear regression using number of formulas to predict reaction times. A significant regression equation is found ($F(1,189) = 432.6, p < 2.2 \cdot 10^{-16}$), with an R^2 of 0.696. Predicted reaction time in milliseconds is equal to $10401 + 50453 \cdot$(number of formulas).

Discussion

The results in the previous sections show the feasibility of our system as a proof-of-concept. For all turn-taking games with at most binary choices, our system generates cognitive models from strategy formulas, without human intervention. This can greatly speed up research, because cognitive models are often created by hand. These models can be run to obtain reaction times, as shown in our results, as well as other data, such as decisions, gazing behaviour, and neural activity. Our verification experiment shows that between the handmade and generated models, the proportion of reaction times between the myopic and own-payoff strategies are highly similar, and the actions they play are the same. We believe this is due to the similarity in their decision-making processes. For example: in both myopic models the model looks at two payoffs in the game tree, stores them in memory, and compares them to make its decision. The difference in reaction times could be due to the following reason: the automatically generated models are specific models for their respective games, whereas the handmade models are general models. Therefore, the handmade models have to perform extra tests to verify whether certain operations are possible in the current game, and to remember what they have already done. These extra steps cannot be removed: the model cannot function without them. However, the generated models are generated from a strategy formula designed for a particular game, so they can simply perform a sequence of actions. This corresponds to the number of primitive elements required by these models. For example, the automatically generated myopic model uses 39 primitive elements, and the handmade myopic model uses 46 primitive elements.

Nonetheless, our exploratory experiments show that our system can provide predictions such as 'human response times in centipede-like games are a function of the number of game-theoretical solutions in a game', which can be experimentally tested. In the future we plan to extend the automatic translation method to perfect-information games with more-than-binary decision points. Similar translation methods of linking formal logic directly to cognitive modeling and behavioural experiments may be constructed for wider classes of tasks, such as planning and communication protocols.

References

Ghosh, S., Heifetz, A., & Verbrugge, R. (2015). Do players reason by forward induction in dynamic perfect information games? In R. Ramanujam (Ed.), *Proc. 15th conf. theor. aspects rationality and knowledge* (pp. 121–130).

Ghosh, S., Heifetz, A., Verbrugge, R., & De Weerd, H. (2017). What drives people's choices in turn-taking games, if not game-theoretic rationality? In J. Lang (Ed.), *Proc. 16th conf. theor. aspects rationality and knowledge* (pp. 265–284).

Ghosh, S., & Verbrugge, R. (online first). Studying strategies and types of players: Experiments, logics and cognitive models. *Synthese*, *2018*.

Meijering, B., Van Rijn, H., Taatgen, N. A., & Verbrugge, R. (2012). What eye movements can tell about theory of mind in a strategic game. *PLoS ONE*, *7*(9), e45961.

Perea, A. (2012). *Epistemic game theory*. Cambridge UP.

Rosenthal, R. (1981). Games of perfect information, predatory pricing and the chain-store paradox. *Journal of Economic Theory*, *25*(1), 92–100.

Stevens, C. A., Daamen, J., Gaudrain, E., Renkema, T., Top, J. D., Cnossen, F., et al. (2018). Using cognitive agents to train negotiation skills. *Frontiers in Psychology*, *9*.

Taatgen, N. A. (2013). The nature and transfer of cognitive skills. *Psychological Review*, *120*(3), 439–471.