Combining Theory of Mind and Abductive Reasoning in Agent-Oriented Programming

Nieves Montes[1†], Michael Luck[2], Nardine Osman[1], Odinaldo Rodrigues[2] and Carles Sierra[1]

[1]Department of Multi-Agent Systems, Artificial Intelligence Research Institute (IIIA-CSIC), Campus de la UAB, Bellaterra, 08193, Barcelona, Spain.
[2]Department of Informatics, King's College London, Bush House, London, WC2B 4BG, UK.

Contributing authors: nmontes@iiia.csic.es; michael.luck@kcl.ac.uk; nardine@iiia.csic.es; odinaldo.rodrigues@kcl.ac.uk; sierra@iiia.csic.es;
[†]Part of this work was done when the author was on a research visit to the Department of Informatics at King's College London.

Abstract

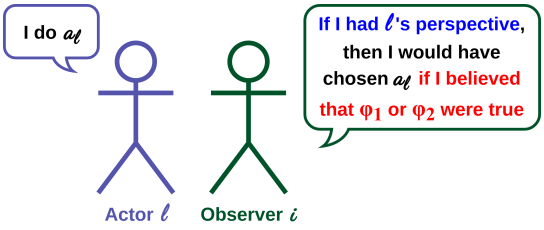
This paper presents a novel model, called TomAbd, that endows autonomous agents with Theory of Mind capabilities. TomAbd agents are able to simulate the perspective of the world that their peers have and reason from their perspective. Furthermore, TomAbd agents can reason from the perspective of others down to an *arbitrary level of recursion*, using Theory of Mind of $\boldsymbol{n}^{\text{th}}$ order. By combining the previous capability with abductive reasoning, TomAbd agents can infer the beliefs that others were relying upon to select their actions, hence putting them in a more informed position when it comes to their own decision-making. We have tested the TomAbd model in the challenging domain of Hanabi, a game characterised by cooperation and imperfect information. Our results show that the abilities granted by the TomAbd model boost the performance of the team along a variety of metrics, including final score, efficiency of communication, and uncertainty reduction.

Keywords: Theory of Mind, abductive reasoning, agent-oriented programming, social AI, Hanabi

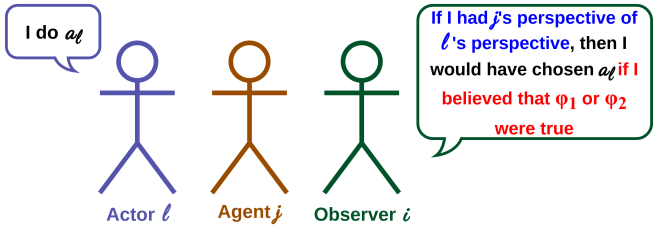
1 Introduction

The emergent field of social AI deals with the formulation and implementation of autonomous agents that can successfully act as part of a larger society, made up of other software agents as well as humans [1, 2]. In human social life, an essential requirement for effective participation is the ability to interpret and predict the behaviour of others in terms of their mental states, such as their beliefs, goals and desires. This ability to put oneself in the position of others and reason from their perspective is called Theory of Mind (ToM) and is closely related to feelings of empathy [3] and moral judgements [4].

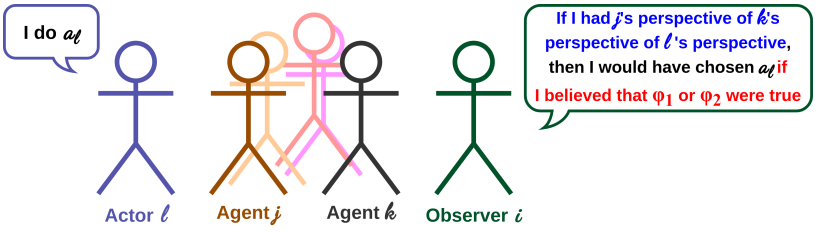
The work presented here starts from the assumption that, just as humans need a functioning ToM, if autonomous software agents are to operate satisfactorily in social contexts, they also need some implementation of the abilities that ToM endows humans with [5]. In particular, in domains where agents have to deal with partial observability, agents can benefit by engaging in the



(a) First-order ToM.



(b) Second-order ToM.



(c) Third-order ToM.

Fig. 1: Outline of the reasoning process captured by the TomAbd agent model.

type of reasoning pictured in Figure 1: agents can infer additional knowledge from observing the actions performed by others and deducing the beliefs that their peers were relying upon to select those actions. This process can be achieved directly as in Figure 1a, where an observer adopts the perspective of an actor to provide an explanation for their action, or through one (Figure 1b) or more (Figure 1c) intermediaries, where the observer adopts the perspective of the actor through an arbitrary number of agents. Hence, agents can use other agents as "sensors" with the purpose of being in a more informed position when it comes to their own decision-making. The backward inference from observations (actions by others) to their underlying motivations is called *abductive reasoning* and, together with ToM, is a central component of the agent model presented here.

The main contribution of this work is the TomAbd agent model, which combines the two capacities mentioned above (Theory of Mind and abduction) to provide the reasoning displayed in Figure 1. This paper builds on a previous, much-reduced, preliminary version [6]. Here, we propose a completely domain-independent model where agents observe the actions of others, adopt their perspective and generate explanations that justify their choice of action. We cover all the steps involved in this reasoning process: from the switch from the agent's perspective to that of a peer's, to the generation, post-processing and update of previous explanations as the state of the system evolves. In addition, we also provide a complementary decision-making function that takes into account the gathered abductive explanations.

We implement the TomAbd agent model in Jason [7], an agent-oriented programming language based on the BDI architecture. Given the functionalities of our model, the ToM capabilities of TomAbd agents are strongly skewed towards the perception step of the BDI reasoning cycle (i.e. upon observation of an action by another agent, generate a plausible explanation for it). Nonetheless, we open up an avenue to introduce ToM reasoning into the deliberation step of the BDI cycle as well through a complementary decision-making function.

Furthermore, we have applied the TomAbd agent model to Hanabi, a cooperative card game that we use as our benchmark. We clearly indicate the specific domain-dependent choices necessary in this application, that need not be shared for other domains. We analyse the model's performance on a number of metrics, namely absolute team score and information gain and value. Our assessment quantifies the gains that can be unequivocally attributed to the ToM abilities of the agents.

This paper is organised as follows. In Section 2 we provide the necessary background on Theory of Mind, abductive reasoning and the Hanabi game. The central contribution of this paper, the TomAbd agent model, is exposed in detail in Section 3. Then, in Section 4 we cover some issues related to the implementation and potential customisations of the model components. Section 5 presents the performance results of the TomAbd agent model applied

to the Hanabi domain. Finally, Section 6 compares our work with related approaches, and we conclude in Section 7.

2 Background

2.1 Theory of Mind

The first building block of the TomAbd agent model is Theory of Mind (ToM). Broadly defined, ToM is the human cognitive ability to perceive, understand and interpret others in terms of their mental attitudes, such as their beliefs, emotions, desires and intentions [8]. Humans routinely interpret the behaviour of others in terms of their mental states, and this ability is considered essential for language and participation in social life [3].

ToM is not an innate ability. It is an empirically established fact that children develop a ToM at around the age of 4 [9]. It has been demonstrated that around this age, children are able to assign false beliefs to others, by having them undertake the Sally-Anne test [10]. The child is told the following story, accompanied by dolls or puppets: Sally puts her ball in a basket and goes out to play; while she is outside Anne takes the ball from the basket and puts it in a box; then Sally comes back in. The child is asked where will Sally look for her ball. Children with a developed ToM are able to identify that Sally will look for her ball inside the basket, thus correctly assigning a false belief to the character, that they themselves know to be untrue.

During the 1980s, the ToM hypothesis of autism gained traction, which states that deficits in the development of ToM satisfactorily explain the main symptoms of autism. This hypothesis argues that the inability to process mental states leads to a lack of reciprocity in social interactions [10]. Although a deficiency in the identification and interpretation of mental states remains uncontested as a cause of autism, it is no longer viewed as the only one, and the disorder is now studied as a complex condition involving a variety of cognitive mechanisms [11, 12].

Within philosophy and psychology, two distinct accounts of ToM exist: Theory ToM (TT) and Simulation ToM (ST) [13]. The TT account views the cognitive abilities assigned to ToM as the consequence of a theory-like body of implicit knowledge. This knowledge is conceived as a set of general rules and laws concerned with the deployment of mental concepts, analogous to a theory of folk psychology. This theory is applied inferentially to attribute beliefs, goals, and other mental states and predict subsequent actions.

In contrast, the ST account views the predictions of ToM not as a result of inference, but through the use of one's own cognitive processes and mechanisms to build a model of the minds of others and the processes happening therein. Hence, to attribute mental states and predict the actions of others, one imagines oneself as being in the other agent's position. Once there, humans apply their own cognitive processes, engaging in a sort of simulation of the minds of others. This internal simulation is very closely related to empathy, since it essentially consists of experiencing the world from the perspective of someone

else. In this work, we adhere more closely to the ST account than to the TT one, as we view the former as having a clearer path to becoming operational. In our TomAbd model, agents simulate themselves to be in the position of another, and then apply abductive reasoning (covered in Section 2.2) to infer their beliefs.

Formally, ToM statements can be expressed using the language of epistemic logic, which studies the logical properties of knowledge, belief, and related concepts [14, 15]. The belief of agent i is expressed using modal operator B_i. Although modal operators also exist for other mental states such as desires and intentions [16], we focus on B, since the ToM abilities of our agent model are manifested by having the agent's own beliefs replaced by an estimation of the beliefs of others. Then, the statement $B_i\phi$ is read as "agent i believes that ϕ".

ToM statements can be expressed by nesting the previous beliefs about the state of the world. Therefore, statement $B_iB_j\phi$ is read as "i believes that j believes ϕ". This corresponds to a *first-order* ToM statement from the perspective of i. Subsequent nesting results in statements of higher order. For example, $B_iB_jB_k\phi$ is read as "i believes that j believes that k believes ϕ", a *second-order* ToM statement. This recursion can be extended down to an arbitrary nesting level. In general, an n-th order ToM statement is expressed as $B_iB_{j_1} \dots B_{j_{n-1}}B_{j_n}\phi$ and is read as "i believes that j_1 believes ... that j_{n-1} believes that j_n believes ϕ". The psychologist Corballis argued that, in fact, the ability to think recursively beyond the first nesting level, as in ToM statements of second order and beyond, is a uniquely human capacity that sets us apart from all other species [17, 18].

Within AI, implementations of ToM are often categorised under the umbrella of techniques for *modelling others* [19]. In the majority of cases, these techniques are applied to competitive domains, where they are referred to as *opponent modelling* [20, 21]. ToM for autonomous software agents has so far been developed in a somewhat fragmented fashion, with every camp within the field implementing it according to their own techniques and methods.

In machine learning, prominent work by Rabinowitz et al. [22] has modelled ToM as a meta-learning process, where an architecture composed of several deep neural networks (DNN) is trained on past trajectories of a variety of agents, including random, reinforcement learning (RL) and goal-directed agents, to predict action at the next time-step. The component of the architecture most related to ToM is the *mental net*, which parses trajectory observations into a generic mental state embedding. It is not specified what kind of mental states (i.e. beliefs or goals) these embeddings represent. In contrast, Wang et al. [23] also use an architecture based on DNNs for reaching consensus in multi-agent cooperative settings. Their *ToM net* explicitly estimates the goal that others are currently pursuing based on local observations. Finally, an alternative approach by Jara-Ettinger [24] proposes to formalise the acquisition of a ToM as an inverse reinforcement learning (IRL) problem. However, these approaches have drawn some criticism for their inability to mimic the actual operation of the human mind, as the direct mapping from

past to future behaviour bypasses the modelling of relevant mental attitudes, such as desires and emotions [25]. By contrast, in our work ToM is used to derive explicit beliefs. We leave the expansion of the model to include other mental states, such as desires and intentions, for future work.

ToM approaches have also been investigated from an analytical game theoretical perspective. De Weerd et al. [26, 27] show that the marginal benefits of employing ToM diminish with the nesting level in competitive scenarios. In particular, while first-order and second-order ToM present a clear advantage with respect to opponents with ToM abilities of lower order (or no ToM capacity at all), the benefits of using higher-order ToM are outweighed by the complexity it entails. The same authors also prove that high-order ToM is beneficial in dynamic environments, with the magnitude of the benefits increasing with the uncertainty of the scenario [28]. It is therefore important to devise techniques that attempt to measure the information gained through the addition of ToM of any order, a concern also considered in this paper.

Finally, symbolic approaches to ToM have studied the effects of announcements on the beliefs of others and the ripple-down effects on their desires and the actions they motivate in response, for the purposes of deception and manipulation [29, 30].

2.2 Abductive Logic Programming

The second main component of the TomAbd agent model is abductive reasoning. Abduction is a logical inference paradigm that differs from traditional deductive reasoning [31]. Classical deduction makes inference following the *modus ponens* rule: from knowledge of ϕ and of the implication $\phi \rightarrow \psi$, ψ is inferred as true. In contrast, abduction makes inferences in the opposite direction: from knowledge of the implication $\phi \rightarrow \psi$ and the *observation* of ψ, ϕ is inferred as a possible *explanation* for ψ.

Hence, instead of inferring conclusions deductively, abduction is concerned with the derivation of hypothesis that can satisfactorily explain an observed phenomenon. For this reason, abduction is broadly defined as "inference to the best explanation" [32], where the notion of *best* needs to be specified by some domain-dependent optimality criterion. Abduction is also distinct from the inference paradigm of *inductive reasoning* [33]. While induction works on a body of observations to derive a general principle, explanations inferred in abductive reasoning consist of *extensional* knowledge, i.e. knowledge that only applies to the domain under examination.

In the context of logic programming, the implementation of abductive reasoning is called Abductive Logic Programming (ALP) [34, 35], defined as follows.

Definition 1 An *Abductive Logic Programming theory* is a tuple $\langle T, A, IC \rangle$, where:

- T is a *logic program* representing expert knowledge in the domain;

- A is a set of ground *abducibles* (which are often defined by their predicate symbol), with the restriction that no element in A appears as the head of a clause in T; and
- IC is a set of *integrity constraints*, i.e. a set of formulas that cannot be violated.

Then, an abductive explanation is defined as follows.

Definition 2 Given an ALP theory $\langle T, A, IC \rangle$ and an observation Q, an *abductive explanation* Δ for Q is a subset of abducibles $\Delta \subseteq A$ such that:

- $T \cup \Delta \models Q$; and
- $T \cup \Delta$ verifies IC.

The verification mentioned in Definition 2 can take one of two views [34]. First, the stronger *entailment* view states that the extension of T with explanation Δ needs to derive the set of constraints, $T \cup \Delta \models IC$. Second, the weaker *consistency* view states that it is enough for the extended logic program not to violate IC, i.e. $T \cup \Delta \cup IC$ is satisfiable, or $T \cup \Delta \not\models \neg IC$. In this work, we adhere to the latter view. We do not model integrity constraints directly but rather their negation. We introduce into the agent program formulas that should *never* hold true through special rules called *impossibility clauses*. More details on this are provided in Section 3.1. Taking the consistency position allows us to work with incomplete abductive explanations that need not complement the current knowledge base to the extent that IC can be derived, but that nonetheless provide valuable information.

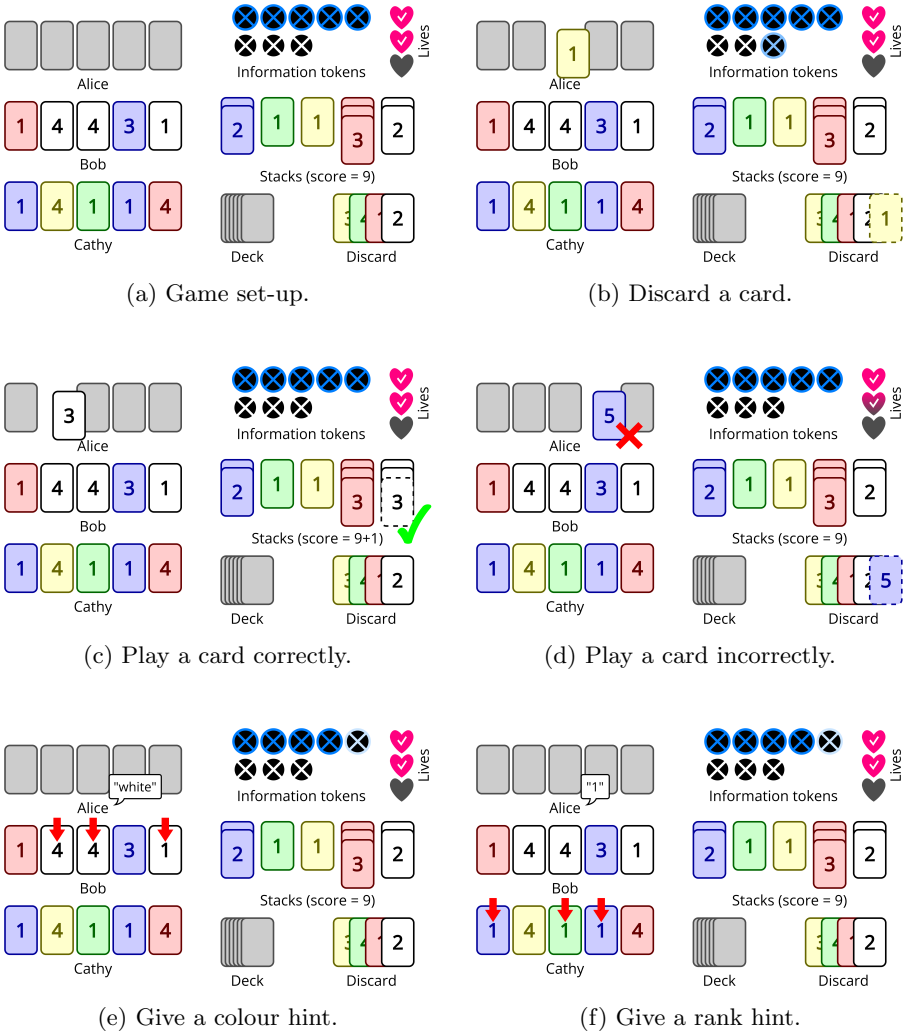
In practice, most existing ALP frameworks compute abductive explanations using some extension of classical Selective Linear Definite (SLD) clause resolution, or its negation-as-failure counterpart SLDNF [36–39]. The current state of the art integrates abduction in Probabilistic Logic Programming (PLP), where the optimal explanation is considered to be the one that is compatible with the constraints and simultaneously maximises the joint probability of the query and the constraints [40].

The purpose of computing abductive explanations is to expand an existing knowledge base KB, which may or may not correspond to the logic program T used to compute explanation δ in the first place. During knowledge expansion, which occurs one formula at a time, the following four scenarios may arise [34].

1. The new information can already be derived from the existing explanation, $KB \cup \delta \equiv KB$, and hence Δ is *uninformative*.
2. KB can be split into two disjoint parts, $KB = KB_1 \cup KB_2$, such that one of them, together with the new information, implies the second, $KB_1 \cup \delta \models KB_2$. In the worst case, the addition of δ renders a part of the original knowledge base redundant.

3. The new information δ violates the logical consistency of KB. To integrate the two, it is necessary to modify and/or reject a number of the assumptions in KB or in δ that lead to the inconsistency.
4. δ is independent and compatible with KB. This is the most desirable case, as δ can be assimilated into KB in a straightforward manner.

In the TomAbd agent model, we deal with scenarios 1 and 3 through the post-processing of the generated abductive explanations by the explanation revision function (ERF). Essentially, uninformative explanations (scenario 1)



(a) Game set-up.

(b) Discard a card.

(c) Play a card correctly.

(d) Play a card incorrectly.

(e) Give a colour hint.

(f) Give a rank hint.

Fig. 2: Basic set-up for the Hanabi game and actions that can be performed.

as well as explanations that violate the integrity of the current belief base (scenario 3) are discarded. More details are provided in Section 3.3.

Hence, the addition of abductive explanations does not affect the correctness of KB, but it may affect its efficiency. The addition of δ into the knowledge base may subsume some information already there, as anticipated by scenario 2. However, in the TomAbd model, we do not check whether a new explanation renders part of the knowledge base redundant. We work with *dynamic* belief bases, which change as agents update their perceptions of the environment. When the system evolves and an agent's perception of it changes, some abductive explanations currently in the belief base need to be dropped because they are no longer correct, or they are now redundant. This operation is performed by the explanation update function (EUF), covered in Section 3.3. If, due to the addition of δ, a part of the belief base had been discarded, it would raise the issue of whether it needs to be recovered once the explanation that caused it to become irrelevant is dropped. We bypass this question by retaining all of the belief base upon adding an explanation, provided that this explanation has previously passed all the redundancy and consistency checks.

2.3 The Hanabi Game

In this paper, we use the Hanabi game as a running example for the presentation of the TomAbd agent model and to evaluate its performance. Hanabi has been by other AI researchers as a testbed to test techniques for multi-agent cooperation [41, 42]. Hanabi is an award-winning[1] card game, where a team of two to five players work together towards a common objective. The goal of the team is to build stacks of cards of five different colours (blue, green, yellow, red and white), with the stacks composed of a card of rank 1, followed by a card of rank 2, and so on, until the stack is completed with a card of rank 5. A typical setup of an ongoing Hanabi game appears in Figure 2a.

At the start of the game, players are handed four or five cards, depending on the size of the team. Players place their cards in a way such that everyone except themselves can see them. For example, the setup in Figure 2a is drawn from the perspective of player Alice, who cannot see her own cards but has access to Bob's and Cathy's cards. Initially, no stack has any card on it (their size is 0). Additionally, eight information tokens (the round blue and black chips in Figure 2) and three live tokens (the heart-shaped chips in Figure 2) are placed on the table.

Players take turns in order, one at a time, in which they must perform one of three actions. First, they can *discard* a card (Figure 2b). Here, the player picks a card from their hand and places it in the discard pile, which is observable by everyone. By doing so, they recover one spent information token (which is spent by giving hints) and replace the vacant slot in their hand with a card drawn from the deck. A player cannot discard a card if there are no information tokens to recover.

[1] https://www.spiel-des-jahres.de/en/games/hanabi/

Second, players can *play* cards from their hand. They pick a card and place it on the stack of the corresponding colour. Players need not state in which stack they are going to play their card before they do so. In other words, they are allowed to play "blindly". There are two possible outcomes to this move. The card is correctly played if its rank is exactly 1 unit over the size of the stack of the card's colour. For example, in Figure 2c Alice plays her white 3 card on the white stack, which has size 2 (i.e. there is a white 1 card at the bottom and a white 2 card in top of it). After a card is played correctly, the team score is increased by 1 unit (the score corresponds to the sum of the ranks at the top of each stack). Moreover, if a player correctly places a card of rank 5 and therefore completes one stack, one information token is recovered for the team, assuming there are some tokens left to recover. Finally, the player replaces the gap in their hand with a card from the deck.

The card is incorrectly played if the rank does not match the size of the stack plus 1. For example, in Figure 2d Alice attempts to play a blue 5 while the blue stack has size of 2. If this happens, the player places the card they attempted to play in the discard pile, and replaces it with a new card from the deck. Furthermore, the whole team loses one of their life tokens.

Third, players can *give hints* to one another about the cards they hold. Hints are publicly announced, i.e. everyone hears them. Players can hint to one another about the colour (Figure 2e) or rank of their cards (Figure 2f). In order to give a hint, the moving player must spend one information token. The team must have at least one information token, which is spent when the hint is given. When players give hints to others, they must indicate all of the receiver's cards that match the colour or rank being hinted. For example, in Figure 2e, Alice has to tell Bob where all of his white cards are. Alice is not allowed to tell Bob only the colour of a card in a single slot if he has other cards of the same colour. Analogously, in Figure 2f, Alice tells Cathy which of her cards have rank 1, not mentioning their colour, regardless of any previous hints.

There are three possible ways in which a game of Hanabi might end. First, the players might manage to complete all of the stacks up to size 5, hence finishing the game with the maximum score of 25. Second, the team might lose all three life tokens. In this case, immediately after losing the third life token, the game finishes with the minimum score of 0. Third and last, after a player has drawn the final card from the deck, all participants take one more turn. After that, the game finishes with score equal to the sum of the size of the stacks.

The Hanabi game has three features that make it particularly interesting to test techniques for modelling others. This has led some researchers to point to Hanabi as the next great challenge to be undertaken by the AI community [41]. The first feature is the purely cooperative nature of the game, since all participants have a common goal, which is to build the stacks as high as possible. Consequently, players can benefit from understanding the mental state of others, such as their intentions with respect to their cards, or the short-term goals

they want to achieve during the course of a game. Additionally, the effectiveness of the developed approaches can be experimentally assessed through the final score.

Second, players in Hanabi have to cope with *partial observability* (or *imperfect information*, the preferred term in the game theory community), as players can see everyone else's cards but not their own. To cope with this, players provide information to one another through hints. There are two facets to these hints. One is the explicit information carried by the hint, i.e. the colour or rank of the cards directly involved. The other facet is the additional implicit information that can be derived from understanding the intention of the player making a move when they provide a hint.

To understand this second facet, consider the situation displayed in Figure 2a. It is Alice's turn to move, and she decides to give a colour hint to Cathy, pointing to her rightmost card as being the only red card she has. In principle, Cathy now only knows that her rightmost card is red, and all others are not. However, Cathy may be able to understand that Alice would only provide such a hint if she wanted her to play that card, and since it is red and the red stack has size 3, Cathy's card must be a red 4. Cathy can draw such a conclusion from the observation of the current state of the game, and an assumption about the strategy that Alice is following. In the TomAbd agent model, this implicit information is identified with the abductive explanations that agents are able to generate by taking the perspective of the player making the move.

Finally, the third interesting feature of Hanabi is the fact that the sharing of information is quantified through discrete tokens that must be managed as a collective resource. Agents must manage the number of hint tokens available altogether, by balancing the need to provide a hint in the current state of the game versus discarding a card to recover a token that then becomes available for another hint.

Previous work on autonomous Hanabi-playing agents has followed one of two approaches: rule-based and reinforcement learning (RL) agents. Rule-based Hanabi bots [43–47] play following a set of pre-coded rules. In contrast, RL bots [41, 48–50] apply single-agent or multi-agent RL techniques to learn a policy for the game. Sarmasi et al. [51] have compiled a database of Hanabi-playing agents developed so far.

Our TomAbd agent models relies on a pre-coded strategy to decide what action to take next and hence aligns more closely with the rule-based approach. However, our agent model is agnostic with respect to the specifics of the strategy that the agent follows. In contrast, previous work on rule-based agents for Hanabi [43–47] has focused on the details of the developed strategies. Also, unlike both rule-based and RL agents, our agent model is domain-independent, and it is applied to Hanabi as a test case. The type of reasoning that TomAbd agents engage in is general but can be useful for this particular game.

Autonomous agents for Hanabi can be evaluated in three different settings: self-play, where all the participants of the team follow the same approach

and strategy; cross-play, where teams are composed of heterogeneous software agents; and human-play, where teams include human players. The majority of the current research on Hanabi AI evaluates performance during self-play, as we do in this paper. In self-play, RL agents outperform rule-based agents, with the former routinely achieving average scores of around 23 points, while the latter struggle to break into 20 points for the average score. The current state-of-the-art for Hanabi AI combines both RL and rule-based techniques, and produced an average score of 24.6 in self-play [49]. To achieve that, first, one agent learns a game-playing policy while all other team members follow the same pre-coded strategy. Second, all agents use multi-agent learning, where they perform the same joint policy update after every iteration, if feasible. If not, they fall back on the same set of pre-coded rules.

Although RL agents display superior performance in self-play, they require a computationally intensive learning process. Additionally, in a recent survey [42] several types of rule-based or RL agents were paired with human players, forming teams of 2. Despite there being no statistically significant difference in game score between rule-based and RL teammates, humans perceived rule-based agents as more reliable and predictable, while expressing feelings of confusion and frustration more often when paired with RL teammates.

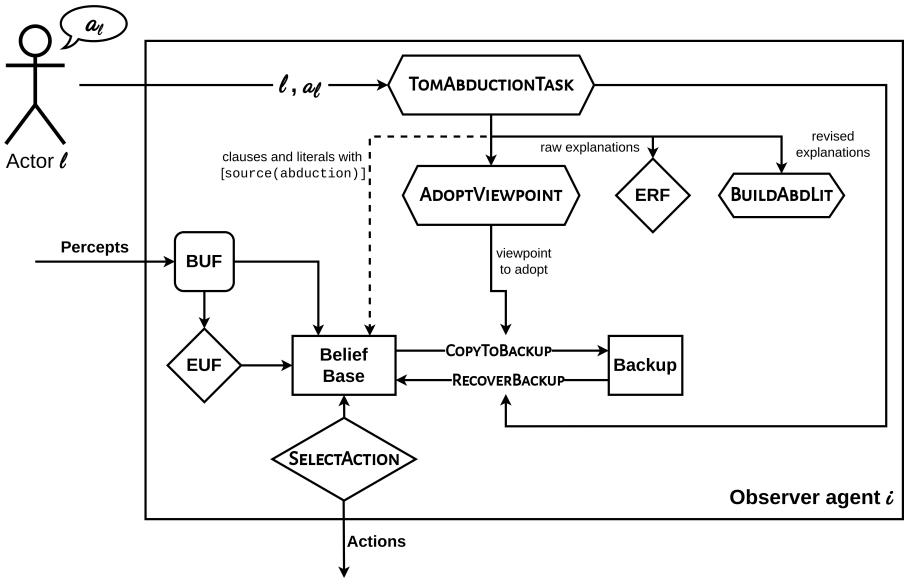


Fig. 3: Architecture of the TomAbd agent model.

3 Agent model

In the current section, we detail the TomAbd agent model, which constitutes the core of this work. First, we outline the agent architecture, its components and introduce some necessary notation. Later, we explain how these components operate.

3.1 Preliminaries

TomAbd is a symbolic, domain-independent agent model with the ability to adopt the point of view of fellow agents, down to an arbitrary level of recursion. Consider the traditional multi-agent setting, where a set of agents $\mathcal{A} = \{i, j, k, \ldots\}$ operate in a shared environment. For the remainder of Section 3, the explanations are presented from the perspective of an arbitrary observer agent i; i.e. we will be considering the cognitive processes that i autonomously undertakes when it observes its fellow agents taking actions.

The main components of the TomAbd agent model are presented in Figure 3. Rectangles represent belief base (BB) data structures. Hexagons represent immutable functions, that are not customisable. Diamonds represent functions for whom only default implementations are provided, and that allow users to customise them according to their application's needs. The rounded square for BUF corresponds to the *belief update function*, a common functionality for situated agents. We do not define this function in our work, but tailor the default BUF method in our language of choice to include some operations on the gathered abductive explanations. Details on this are provided in Section 4.

The agent architecture is composed of the main BB data structure which contains the logic program that the agent is currently working with, plus a backup to store the agent's own beliefs when switching to another agent's perspective. At all times, the BB contains a logic program: a set of ground literals representing facts about the world and a set of rules representing relationships between literals. We denote by T_i the logic program of agent i; i.e. the content of their BB at initialization time. T_i is composed of the following components.

1. **Percepts** are ground literals that represent the information that the agent receives from the environment. Incoming percepts update the BB according to the *belief update function* (BUF in Figure 3). For example, in a situation of Hanabi like the one displayed in Figure 2, agent Alice would receive the following percepts:

   ```
   has_card_colour(bob,4,blue)
   has_card_rank(bob,4,3)
   ```

   These indicate that Alice observes Bob having a card of colour blue and rank 3 in his fourth slot (counting from the left). We assume, implicitly, that the agents are limited by partial observability, meaning that they do not perceive all the information there is to know about the environment. In general, different agents will have access to different parts of the environment, and will receive different percepts. In the Hanabi game in particular,

players do not *a priori* know about their own cards or about the order of cards in the deck.

2. **Domain-related clauses** are traditional logic programming rules that establish relationships between facts in the domain. For example, the following clause expresses that, in Hanabi, a card of colour `C` and rank `R` is playable if the size `Sz` of the corresponding stack is one unit below the rank of the card.

```
playable(C,R) :- colour(C), rank(R), stack(C,Sz), Sz=R-1.
```

3. **Impossibility clauses** have atom `imp` as their head and whose body contains literals that cannot hold simultaneously true. They capture the constraints of the domain, if there are any. For example, in the Hanabi game, the following clause states that a player `P` cannot have cards of two different colours, `C1` and `C2`, in the same slot `S`.[2]

```
imp :-
    player(P), slot(S), colour(C1), colour(C2),
    has_card_colour(P,S,C1), has_card_colour(P,S,C2), C1\== C2.
```

   As stated in Section 2.2, we adopt the *consistency view* when it comes to verifying the expansion of the belief base with an abductive explanation. To incorporate integrity constraints into an agent's program, we need a mechanism that triggers an exceptional event when one or several constraints are violated. This is precisely the role of the impossibility clauses.

   To clarify, consider an impossibility clause `imp :- Conj`. The conjunction `Conj` in its body corresponds to a formula that should never hold true. In other words, its negation ¬`Conj` is equivalent to a traditional integrity constraint IC that can never be violated. Therefore, the derivation of `imp` indicates that IC has been violated. To avoid this, the generated abductive explanations undergo post-processing operations where they are filtered out if their expansion into the program causes the derivation of `imp`.

4. **Theory of Mind clauses** are rules that are essential to the agent's cognitive ability to put itself in the shoes of others. They function as a meta-interpreter on the agent's current program to generate an estimation of another agent's program. ToM clauses have the literal `believes(Ag,F)` as their head, to express the fact that agent i believes that agent `Ag` knows about some fact `F`. In the Hanabi domain, the following ToM clause indicates that agent i believes that player `Agj` can see the card that a third player `Agk` has in their `S`-th slot, and, in particular, `Agj` can observe its colour `C`.

```
believes(Agj,has_card_colour(Agk,S,C)) :-
    player(Agj), player(Agk), slot(S), colour(C),
    has_card_colour(Agk,S,C), Agj\==Agk.
```

[2]Note that Jason, our implementation language of choice, does not include any default mechanisms to check the consistency of an agent's BB, i.e. an agent may simultaneously believe b and $\sim b$. It hence becomes the responsibility of the agent developer to implement, if needed, additional mechanisms to avoid such inconsistencies, which the TomAbd agent model achieves through the introduction of impossibility clauses.

5. **Abducible clauses** have literal `abducible(F)` at their head, and express what missing beliefs can potentially be added to agent i's BB to obtain a more detailed representation of the state of the system. The definition of the literals that may be missing from an agent's BB is a domain-dependent component of the program. For example, in the Hanabi domain, the following abducible clause indicates that, from the viewpoint of i, a player `P` may have, in their `S`-th slot, a card of colour `C` if i does not already hold a belief about the colour of the card in `S`, nor does i explicitly hold a belief explicitly indicating that `P` does not have a card of colour `C` in `S`.[3]

```
abducible(has_card_colour(P,S,C)) :-
    player(P),slot(S),colour(C),
    not has_card_colour(P,S,_),
    not ~has_card_colour(P,S,C).
```

6. **Action selection clauses** are a set of rules with head `action(Ag,Act) [priority(`n`)]` that indicate the pre-conditions for agent `Ag` to select and execute action `Act`. These clauses correspond to agent i's beliefs about the other agents' strategies (for instances where `Ag` $= j, j \neq i$) as well as, potentially, agent i's own strategy (for instance when `Ag` $= i$). The head is annotated with a `priority(`n`)` literal, where n is a number (any number, not necessarily an integer). These priorities state in which order the action selection clauses should be considered when they are queried. Details about the action selection are provided in Section 3.4.

   As an example for the Hanabi domain, the following clause indicates that a participant `P` should play their card in slot `S` if it is of a playable colour `C` and rank `R`.

```
action(Ag,play_card(S)) [priority(3.0)] :-
    player_turn(Ag), slot(S),
    has_card_colour(Ag,S,C), has_card_rank(Ag,S,R),
    playable(C,R).
```

   The action selection clauses are used by TomAbd agents to compute abductive explanations from the observation of actions by other agents. Hence, it is compulsory that they capture agent i's beliefs about the strategy agent $j \neq i$ is following. Nonetheless, the TomAbd model is flexible concerning whether such action selection clauses also implement agent i's own strategy. The action selection function presented in Section 3.4 certainly provides an avenue to use action selection clauses during the agent's own practical reasoning. However, this is a complement to the TomAbd model (whose focus is on the generation and maintenance of abductive explanations using ToM) rather than a fundamental component.

So far, we have presented the components of the logical program of a TomAbd agent. Now, we move on to explain how they are utilised. The distinguishing feature of our agent model is the ability to put themselves in the shoes of others. For example, when engaging in first-order ToM (recursive

[3] We distinguish between strong negation (`~Fact`) and negation as failure (`not Fact`). In epistemic logic notation, they are expressed as $B_i[\sim \phi]$ and $\sim B_i\phi$, respectively.

level 1), agent i changes their perception of the world to the way in which they believe that some other agent j is perceiving it. In epistemic logic notation, these are the beliefs denoted by $B_i B_j \phi$. In other words, in an attempt to perceive the world how i believes j is perceiving it, agent i's BB changes to $BB_i^j = \{\phi \mid \texttt{believes}(j, \phi)\}$, where BB_i^j denotes i's estimation of j's BB given i's current logic program.

However, the TomAbd agent model is not limited to first-order ToM. It can, in fact, switch its perception of the world to that of another agent *down to an arbitrary level of recursion*. For example, agent i may want to view the world in the way that they believe j believes that k is perceiving it. This corresponds to second-order ToM and is expressed as $B_i B_j B_k \phi$ in epistemic logic notation. In particular, i may want to estimate j's estimation of itself. This is equivalent to the previous case $B_i B_j B_k \phi$ with $i = k$, $B_i B_j B_i \phi$.

The nesting exposed in the previous paragraph can be extended to an arbitrary level of recursion: agent i attempts to view the world how it believes that j believes ... that k believes that l views it. This is denoted by $B_i B_j \ldots B_k B_l \phi$. We define the sequence of agent perspectives $[j, \ldots, k, l]$ recursively adopted by i as a *viewpoint*:

Definition 3 For agent i, a *viewpoint* is an ordered sequence of agent designators $[j, \ldots, k, l]$ where there are no two consecutive equal elements and the first element is different from i.

Hence, when we talk about agent i *adopting viewpoint* $[j, \ldots, k, l]$ we mean the process by which agent i switches its own perspective of the world by the one it believes that j believes ... that k believes that l has. To do this, agent i has to modify its own program T_i, contained in its main BB, by the estimation that it can build of j's estimation ... of k's estimation of l's program. This new program will, in general, indeed be an *estimation* since agents have access to (possibly) overlapping but different features of the environment. We denote this estimated program by $T_{i,j,\ldots,k,}$ and define it as follows:

Definition 4 Given agent i with logic program T_i, *i's estimation of viewpoint* $[j, \ldots, k, l]$ is a new logic program $T_{i,j,\ldots,k,l}$:

$$T_{i,j,\ldots,k,l} = \{\phi \mid T_{i,j,\ldots,k} \models \texttt{believes}(l, \phi)\} \tag{1}$$

Equation (1) indicates that, in order to estimate the BB of the next agent whose perspective is to be adopted, the agent must query its current BB to find all the ground literals that, according to the ToM rules, the next agent knows about. Therefore, agent i must substitute their current program by the set of unifications to the second variable in `believes(Ag,F)`.

The operationalisation of Definition 4 is presented in function AdoptViewpoint, Algorithm 1. It takes as its only argument a viewpoint as defined in Definition 3. Given this viewpoint, agent i adopts it, first, by saving

Algorithm 1 Function AdoptViewpoint(vp)

Input: vp ($[j, \ldots, k, l]$), a viewpoint according to Definition 3.
Result: agent i substitutes T_i by $T_{i,j,\ldots,k,l}$ in their BB.

1: CopyBBToBackup()
2: **for** p in vp **do**
3: $\quad BB_i^p \leftarrow \{\phi \mid BB \models \texttt{believes}(p, \phi)\}$
4: $\quad BB \leftarrow BB_i^p$
5: **end for**

a copy of its own BB in the backup. Then, i queries the ToM clauses with the next agent whose perspective is to be estimated as their first argument. The result of this operation becomes agent i's new BB, and they move on to the next iteration.

3.2 The TomAbductionTask Function

Function AdoptViewpoint captures the n^{th}-order Theory of Mind capabilities in the TomAbd agent model, for arbitrary integer value of n. However, the purpose of switching one's perspective is to be able to reason from the point of view of another agent. Therefore, it is not enough for i to invoke AdoptPerspective. It should, once the switch has occurred, infer the motivation for the actions taken by the other. This reasoning process is implemented in the core function of the TomAbd agent model, TomAbductionTask, in Algorithm 2.

TomAbductionTask takes three arguments as input: an observer viewpoint, an acting agent l and the action l took a_l. The last two are straightforward to understand. The observer viewpoint is a list as defined in Definition 3. It indicates what ToM order i is engaging in, and through which other agents it is estimating the perception that the actor l has of the world. For example, suppose i would like to understand why l chose a_l directly. In this case, i observes its peer's action from its own perspective. The observer viewpoint would, in this case, correspond to the empty list ("[]"). However, i might want to understand why a third agent j thinks that l made their choice. Then, i is observing l's action through j, and hence the observer viewpoint is $[j]$. This viewpoint can be subsequently extended to any desired level of recursion. For example, agent i may want to estimate the impression that actor l thinks they are making on agent j when executing a_l. This corresponds to observer viewpoint $[l, j]$.

The first step of TomAbductionTask (Lines 1 and 2) is to build the *actor's viewpoint* by simply appending actor agent l to the observer viewpoint and to adopt it by calling AdoptPerspective. Now, agent i is in a position to reason from the perspective of the actor, possibly through a number of intermediate observers. In the simplest case, agent i is switching its logical program T_i to the program it estimates actor l to be working with, i.e. $T_{i,l}$. This case corresponds to agent i engaging in *first-order* ToM at the time of adopting the actor's viewpoint. Alternatively, agent i may switch its program

Algorithm 2 Function TomAbductionTask($obsVp$, l, a_l)

Input: $obsVp$, observer viewpoint.
l, acting agent.
a_l, action.

Output: $\mathbf{\Phi'_{act}}$, `actLit`, $\mathbf{\Phi'_{obs}}$, `obsLit`: explanations $\mathbf{\Phi'}_{\{\cdot\}}$ that justify the election of action a_l by actor agent l, from the actor's and the observer's viewpoint, respectively; and the literals $\{\cdot\}$`Lit` containing those explanations in a format suitable to be added to a logic program.

1: $actVp \leftarrow obsVp$.Append(l)
2: AdoptViewpoint($actVp$)
3: $BB \leftarrow BB \cup \{$`viewpoint(actor)`$\}$
4: $\mathbf{\Phi} \leftarrow$ Abduce(`action`(l, a_l))
5: $\mathbf{\Phi'_{act}} \leftarrow$ ERF($\mathbf{\Phi}$)
6: `actLit` $\leftarrow$ BuildAbdLit($actVp$, $\mathbf{\Phi'_{act}}$)
7: RecoverBackup()
8: AdoptViewpoint($obsVp$)
9: $BB \leftarrow BB \cup \{$`viewpoint(observer)`$\}$
10: $\mathbf{\Phi'_{obs}} \leftarrow$ ERF($\mathbf{\Phi}$)
11: `obsLit` $\leftarrow$ BuildAbdLit($obsVp$, $\mathbf{\Phi'_{obs}}$)
12: RecoverBackup()
13: **return** $\langle \mathbf{\Phi'_{act}}$, `actLit`, $\mathbf{\Phi'_{obs}}$, `obsLit`$\rangle$

T_i to the program they estimate that j_1 estimates that ... j_{n-1} estimates that j_n is working with, $T_{i,j_1,\ldots,j_{n-1},j_n}$, this time engaging in n^{th}-*order* ToM.

Once the actor's viewpoint has been adopted, the agent uses ALP to generate abductive explanations that justify agent l's action a_l. The ALP theory that the agent uses is composed of its current BB (in the general case, $T_{i,j,\ldots,k,l}$), and the set of abducibles derived from it, which we denote as $A_{i,j,\ldots,k,l}$:

$$A_{i,j,\ldots,k,l} = \{\alpha \mid T_{i,j,\ldots,k,l} \models \texttt{abducible}(\alpha)\} \tag{2}$$

The set of plausible abductive explanations is computed by function Abduce in Line 4 of Algorithm 2, using the set of abducibles defined in Equation (2). The pseudocode for this function is not provided, as it does not constitute any technical innovation. The input to this function is the query $Q = \texttt{action}(l, a_l)$. The Abduce function consists of an abductive meta-interpreter, based on classical SLD clause resolution with a small extension. To compute abductive explanations, this meta-interpreter attempts to prove the query Q as a traditional goal in SLD clause resolution. However, when it encounters a sub-goal that is not provable, before failing the query, it checks whether this sub-goal can be unified to any element in the set of abducibles $A_{i,j,\ldots,k,l}$. If so, the sub-goal is added to the explanation under construction in the branch being currently explored.

Function Abduce backtracks upon failure or completion of the query, just as traditional SLD solvers. Consequently, the output of this function is a set

$\mathbf{\Phi}$ of m potential explanations. At the same time, every element in $\mathbf{\Phi}$ is itself a set of ground abducibles from $A_{i,j,\ldots,k,l}$:

$$\begin{aligned} \mathbf{\Phi} &= \{\Phi_1, \ldots, \Phi_m\}, \text{ where } \Phi_h = \{\phi_{h1}, \ldots, \phi_{hn_h}\} \\ &\text{and } \phi_{hg} \in A_{i,j,\ldots,k,l}, \forall h, g \end{aligned} \tag{3}$$

Once the abductive explanations have been computed, they are first refined through the application of the *explanation revision function*, ERF, in Line 5. Then, they are transformed into a literal, that is, to a format suitable to be added to a logical program, through the BuildAbdLit function in Line 6. Both of these steps are reviewed in detail in the next section.

At this point, the abductive explanations have been computed and post-processed, all from the perspective of the actor, i.e. whilst agent i's BB contains $T_{i,j,\ldots,k,l}$. However, agent i does not derive this information *only* so that it can build a better estimation of the actor's BB. It also reasons about how this information affects beliefs at the *observer's viewpoint level*. Therefore, agent i has to first return to its original program T_i by retrieving it from the backup (Line 7). Then, it adopts the *observer's* viewpoint (Line 9) and perform the same post-processing steps (explanation revision in Line 10 and format transformation in Line 11) from this new perspective. Eventually, agent i recovers its original program T_i from the backup in Line 12.

It should be noted that the TomAbductionTask function does not, by default, add the abductive explanations (or rather, the associated literals generated by BuildAbdLit) to agent i's program T_i (observe the dashed arrow from TomAbductionTask to the BB in Figure 3). Rather, the function returns the revised explanations and their formatted literals. This choice has been made to allow flexibility to potential users. If necessary, users can perform further reasoning and modifications to the returned explanations. For example, agent i can decide whether to append the returned literals to their BB based on some trust metric it has towards the actor.

3.3 Explanation Revision, Assimilation and Update

This section reviews the post-processing operations that are performed on the raw abductive explanations returned by the Abduce function. In the cases where the implementations provided are *defaults*, this is clearly indicated. Details of how these defaults can be overridden are provided in Section 4.

During the execution of TomAbductionTask, two calls are made to the *explanation revision function* (ERF), one from the point of view of the actor and one from the point of view of the observer. The purpose of this function is to refine and/or filter the raw explanations based on the current content of agent i's BB, which is either the estimation of the actor's program $T_{i,j,\ldots,k,l}$ or the estimation of the observer's program $T_{i,j,\ldots,k}$.

The default implementation of the ERF function appears in Algorithm 3 and consists of two steps. First, in Line 3, the agent trims every explanation Φ_h (a set of ground abducibles) to remove *uninformative* atoms. Admittedly,

Algorithm 3 Function ERF($\mathbf{\Phi}$) (*default* explanation revision function)

Input: $\mathbf{\Phi} = \{\{\phi_{11}, \ldots, \phi_{1n_1}\}, \ldots, \{\phi_{m1}, \ldots, \phi_{mn_m}\}\}$, a set of m explanations, each being a set of ground abducibles.

Output: $\mathbf{\Phi'} = \{\{\phi'_{11}, \ldots, \phi'_{1n'_1}\}, \ldots, \{\phi'_{m'1}, \ldots, \phi'_{m'n'_{m'}}\}\}$, a set of m' refined explanations.

1: $\mathbf{\Phi'} \leftarrow \{\}$
2: **for** Φ_h in $\mathbf{\Phi}$ **do**
3: $\quad \Phi'_h \leftarrow \{\phi_{hg} \mid \phi_{hg} \in \Phi_h$ and $BB \not\models \phi_{hg}\}$
4: $\quad$ **if** $BB \cup \Phi'_h \not\models$ `imp` **then**
5: $\quad\quad \mathbf{\Phi'} \leftarrow \mathbf{\Phi'} \cup \{\Phi'_h\}$
6: $\quad$ **end if**
7: **end for**
8: **return** $\mathbf{\Phi'}$

this step only makes a difference when ERF is called from the perspective of the *observer* (Line 10 in Algorithm 2), and not from the perspective of the *actor* (Line 5 in Algorithm 2). The abduction meta-interpreter does not add proven sub-goals to the explanation under construction. Therefore, from the perspective of the actor (where the raw abductive explanations are actually computed), there cannot be uninformative facts in the explanation sets.

The second step is a consistency check (Lines 4 to 6 in Algorithm 3). This check takes in every trimmed explanation and inspects whether it, together with agent i's current BB, entails any impossibility clause. Recall from the discussion in Section 3.1 that the derivation of `imp` is equivalent to an integrity constraint IC being violated. The impossibility clauses that this check considers include both domain-related and impossibility clauses derived from prior executions of TomAbductionTask. If no violation occurs, the explanation is returned as part of the set of revised explanations.

Here, we have only presented a basic ERF implementation that can be customised if needed. For example, the ERF could annotate every explanation Φ_h with an uncertainty metric. Alternatively, it could operate differently depending on whether it is being called while the agent is working under the *actor* or the *observer* point of view. In fact, the belief addition operations in Lines 3 and 9 of Algorithm 2 are there precisely to allow for this possibility. Further details are provided in Section 4.

The set of revised explanations $\mathbf{\Phi'}$ is, like the set of raw explanations $\mathbf{\Phi}$, a *set of sets* of ground abducibles, see Equation (3). Therefore, it is not in a suitable format to be added to agent i's BB, which is a logical program composed of facts and clauses. The conversion from a set of sets to a clause that can be added to a logical program is performed by function BuildAbdLit (short for "build abductive literal") in Algorithm 4.

To understand how this function operates, consider that a (revised) abductive explanation $\mathbf{\Phi} = \{\{\phi_{11}, \ldots, \phi_{1n_1}\}, \ldots, \{\phi_{m1}, \ldots, \phi_{mn_m}\}\}$ can be written

Algorithm 4 Function BUILDABDLIT(vp, $\mathbf{\Phi}$)

Input: vp, a viewpoint
$\mathbf{\Phi} = \{\{\phi_{11}, \ldots, \phi_{1n_1}\}, \ldots, \{\phi_{m1}, \ldots, \phi_{mn_m}\}\}$, a set of (revised) explanations.

Output: Λ, a literal containing the input explanation, in a format suitable to be added to the agent's logical program.

1: $\Lambda \leftarrow$ {`imp [source(abduction)] :-` $(\sim \phi_{11} \mid \ldots \mid \sim \phi_{1n_1})$, ... , $(\sim \phi_{m1} \mid \ldots \mid \sim \phi_{mn_m})$.}
2: $vp' \leftarrow$ REVERSE(vp)
3: **for** p in vp' **do**
4: $\quad \Lambda' \leftarrow$ `believes`(p, Λ)
5: $\quad \Lambda \leftarrow \Lambda'$
6: **end for**
7: **return** Λ

as the following disjunctive normal form (DNF):

$$\mathbf{\Phi} = (\phi_{11} \wedge \ldots \wedge \phi_{1n_1}) \vee \ldots \vee (\phi_{m1} \wedge \ldots \wedge \phi_{mn_m}) \tag{4}$$

The formula in Equation (4) must hold, meaning it has the status of a traditional *IC* discussed in Section 2.2. Therefore, its negation $\neg\mathbf{\Phi}$ must never hold true. If $\neg\mathbf{\Phi}$ is derived from the agent's program, it means that the formula in Equation (4) has been violated, and an exceptional event (i.e. the derivation of `imp`) should be triggered. This observation leads to the use of $\neg\mathbf{\Phi}$ to build a new impossibility clause. This new clause has the same format as the domain-related impossibility clauses presented in Section 3.1 but its head `imp` is annotated with `source(abduction)` to denote that it is not domain-specific but derived from an abductive reasoning process. This step corresponds to Line 1 in Algorithm 4.

Nonetheless, this new impossibility clause does not consider the level of recursion, or, in other words, the *viewpoint*, where the explanation was generated. This information needs to be incorporated in Lines 2 to 6. In summary, if the agent is operating under viewpoint $[j, \ldots, k, l]$, BUILDABDLIT nests the abductive impossibility clause constructed in Line 1 into the following literal:

$$\begin{aligned}\texttt{believes}(j, \ldots, \texttt{believes}(k, \texttt{believes}(l, \{&\texttt{imp [source(abduction)] :-} \\ &(\neg\phi_{11} \mid \ldots \mid \neg\phi_{1n_1}) \texttt{ , } \ldots \texttt{ , } (\neg\phi_{m1} \mid \ldots \mid \neg\phi_{mn_m})\}))\ldots).\end{aligned} \tag{5}$$

Therefore, the next time agent i adopts viewpoint $[j, \ldots, k, l]$, the bare `imp [source(abduction)]` clause will become part of their BB (assuming the user has decided to add it to T_i in the first place).

Finally, there is one last operation performed on the clauses and literals derived from TOMABDUCTIONTASK, which is the *update* of those that have been incorporated into the original BB of the agent, T_i, as new percepts are received. We refer to this operation as the *explanation update function* (EUF).

Algorithm 5 Function EUF (*default* explanation update function)

Result: The agent removes from their BB the abductive literals whose associated explanation is no longer informative.

1: $\Sigma \leftarrow \{\}$
2: **for all** $\langle vp, \mathbf{\Phi}, \texttt{lit} \rangle$ in BB **do**
3: AdoptViewpoint(vp)
4: **if** $BB \models \mathbf{\Phi}$ **then**
5: $\Sigma \leftarrow \Sigma \cup \{\texttt{lit}\}$
6: **end if**
7: RecoverBackup()
8: **end for**
9: $KB \leftarrow KB \setminus \Sigma$

In contrast to the other functions presented in this section, the EUF is not executed within TomAbductionTask, but is called from the *belief update function* (BUF, see Figure 3). The BUF is a standard function of the BDI agent reasoning cycle whose purpose is to update the BB depending on the percepts received from the environment and the messages passed on by other agents. Therefore, upon receiving percepts from the environment, the agent first modifies its ground percept beliefs, and then updates clauses and literals derived from previous executions of TomAbductionTask, if there are any.

The default implementation of EUF appears in Algorithm 5. In it, agent i discards previous abductive explanations if they are deemed to be no longer informative *at the viewpoint at which they were generated*. To do so, the agent loops over all the literals that originated from an abductive reasoning process, denoted by the tuple $\langle vp, \mathbf{\Phi}, \texttt{lit} \rangle$ composed of the viewpoint vp where the explanation $\mathbf{\Phi}$ originated and the associated literal (or clause) `lit` (Line 2). Agent i then adopts viewpoint vp with a routine call to AdoptViewpoint, and checks if explanation $\mathbf{\Phi}$ can be derived from the current BB, $T_{[i|vp]}$.[4] If so, the explanation is deemed to be no longer informative and its associated literal `lit` is added to a removal set.

3.4 Action Selection

The functions presented so far constitute the agent's core cognitive abilities combining Theory of Mind and abductive reasoning. However, the purpose of undergoing all this cognitive work is for agent i to be in a more informed position when it comes to i's own decision-making. To do so, agent i needs to consider the generated abductive explanations when reasoning about which action to perform next. We provide such a function, SelectAction in Algorithm 6, which takes into account *all* the impossibility clauses in the agent's BB, including those coming from the output of TomAbductionTask.

[4] We use Prolog notation for lists $[H \mid T]$, where H is the first element (head) and T is the tail of the list, which is itself another list, possibly empty.

Algorithm 6 Function SelectAction (*default* action selection)

Output: a, an action.

1: **for all** clauses $\mathcal{H}$ with head `action(Ag,Act) [priority(`p`)]`, in descending order of p **do**
2: $\mathcal{H}$.Map(`Ag` $\mapsto i$)
3: $\mathcal{B} \leftarrow \mathcal{H}$.Body()
4: $\Gamma \leftarrow$ SkolemisedAbducibles($\mathcal{B}$)
5: **for all** γ in Γ **do**
6: $\Pi \leftarrow$ Instantiate(γ)
7: $\mathcal{A} \leftarrow \{\}$
8: **for all** π in Π **do**
9: **if** $BB \cup \pi \not\models$ `imp` **then**
10: $a_\pi \leftarrow \underset{a}{\text{argmax}}\{m \mid BB \cup \pi \models$ `action(`i, a`)[priority(`m`)]`$\}$
11: $\mathcal{A} \leftarrow \mathcal{A} \cup \{a_\pi\}$
12: **end if**
13: **end for**
14: **if** $\mathcal{A} = \{a\}, \|\mathcal{A}\| = 1$ **then**
15: **return** a
16: **end if**
17: **end for**
18: **end for**
19: **return** *null*

Similarly to the ERF and EUF, the provided implementation is a basic one, and it is customisable. The user can, for example, reason probabilistically about which action to take next, in case they have associated an uncertainty metric to the generated abductive explanations. The default implementation in Algorithm 6 takes a cautious approach, where an action is only selected if it is the action prescribed by the action selection clauses in all the possible worlds.

Additionally, the SelectAction function presented here is a complement to the other functionalities of the TomAbd agent model, and not a core component of the model. We provide a default querying mechanism to select an action given the action selection clauses and the set of current impossibility constraints. However, the agent developer might decide to use an alternative implementation that, for instance, does not use the action selection clauses to pick the action to execute next, or they might decide to not use the SelectAction function at all. This is enabled by the fact that this function has been wrapped in an internal action (IA) that can be called from within the agent code. More details on this point are provided in Section 4.

Algorithm 6 proceeds as follows. First, in Line 1, it retrieves action selection clauses in descending order of priority, so rules with higher priority take precendence over rules with lower priority. Then, the variable at the first argument in the head is unified with the identity of agent i in Line 2.

Second, the body of the clause is retrieved (Line 3) and the set of *skolemised abducibles* is built. This is done by function SkolemisedAbducibles (whose pseudo-code is not provided) in Line 4. This means that whenever an abducible in the rule body cannot be proven by the agent's BB (i.e. T_i), its free variables are substituted by Skolem constants. In general, one action selection clause will generate several skolemised forms of its abducibles.

Third, the agent searches for all of the potential instantiations of every skolemised form. This corresponds to the call to function Instantiate in Line 6. Again, for every set of skolemised abducibles, there will be, in general, several possible ways of binding their variables. Each of these possible instantiations provides additional beliefs that can partly complement the agent's BB to obtain a more complete view of the current state of the world. However, it is not necessary to complement the agent view to the point of complete observability, just to add enough information to be able to query the action selection clause currently under consideration.

The agent only considers the complete instantiations of abducibles that, together with agent i's BB, do not lead to an impossibility clause. This check takes place in Line 9. For those instantiations that pass the check, agent i queries for the action of maximum priority that is entailed if the grounded abducibles were part of the BB. As we have seen, for this default implementation, action selection clauses with higher priority take precedence over clauses with lower priority. Hence, when querying for actions, the one with the highest priority is returned.

So, every action selection clause (i.e. an iteration of the loop in Line 1) leads to several sets of skolemised abducibles. In its turn, every set of skolemised abducibles (i.e. an iteration of the loop in Line 5) leads to several sets of ground abducibles. If each of these instantiations leads to the same action (Line 14), the SelectAction function returns the action in question and execution continues from the point where the function had been called.

If all the action selection clauses have been processed and no action has been selected, the SelectAction function returns *null*. The user is advised to deal with this situation by including some contingency measure, e.g. use a *default action* when SelectAction return *null*. Further details are provided in Section 4.

To illustrate how the default SelectAction function works, consider the action selection clause provided as an example in Section 3.1:

```
action(Ag,play_card(S)) [priority(3.0)] :-
    player_turn(Ag), slot(S),
    has_card_colour(Ag,S,C), has_card_rank(Ag,S,R),
    playable(C,R).
```

Then, after the execution of Line 3, we have:

$\mathcal{B} \leftarrow$ `player_turn(`i`), slot(S),`
`has_card_colour(`i`,S,C), has_card_rank(`i`,S,R), playable(C,R).`

Now, suppose agent Alice, in the setting of Figure 2a, has the following information in her BB, derived from a hint:

~has_card_rank(i,1,3) [source(hint)]
has_card_rank(i,2,3) [source(hint)]
~has_card_rank(i,3,3) [source(hint)]
has_card_rank(i,4,3) [source(hint)]
~has_card_rank(i,5,3) [source(hint)]

This means that Alice knows that she has cards of rank 3 in her 2nd and 4th slots, and that she has cards of rank different from 3 at all others.

Then, after execution of Line 4, we have:

$\Gamma \leftarrow$ {{has_card_colour(i,1,sk_1), has_card_rank(i,1,sk_2)},
{has_card_colour(i,2,sk_3)},
{has_card_colour(i,3,sk_5), has_card_rank(i,3,sk_6)},
{has_card_colour(i,4,sk_7)},
{has_card_colour(i,5,sk_9), has_card_rank(i,5,sk_{10})}}

where sk_n are Skolem constants.

In addition, suppose that she has in her BB the following IC, derived from a previous execution of TomAbductionTask:

imp [source(abduction)] :-
~has_card_colour(i,2,blue), ~has_card_colour(i,2,white).

This IC would have been derived by BuildAbdLit (Algorithm 4) from the following set of (revised) abductive explanations:

$\mathbf{\Phi}$ = {{has_card_colour(i, 2, blue)}, {has_card_colour(i, 2, white)}}

meaning that, from a previous move by another player, agent i interpreted that they must have either a blue or a white car in the second slot.

Then, when looping through the second element of Γ, in Line 6, the following instantiations will be generated:

$\Pi \leftarrow$ {{has_card_colour(i,2,blue)},
{has_card_colour(i,2,green)},
{has_card_colour(i,2,yellow)},
{has_card_colour(i,2,red)},
{has_card_colour(i,2,white)}}

Of all the instantiations in Π, only two are compatible with the previous IC:

{has_card_colour(i,2,blue)}, {has_card_colour(i,2,white)}

When querying for which action to select, the two previous instantiations will lead to play_card(2) (due to the action clauses with priority 3.0), and this will be the return value of the function SelectAction.

4 Implementation

The agent model presented in Section 3 has been implemented in Jason [7], an agent-oriented programming language based on the BDI architecture. Jason implements and extends the abstract AgentSpeak language [52], offering a wide range of features and options for customisation. To utilize the TomAbd in their projects, the user is required to have prior knowledge on the Jason programming language [7, Chapter 3], the basics of the Jason reasoning cycle [7,

```
+trigger : context
    <- ... ;
    tomabd.agent.tom_abduction_task(
        +ObsVp,                    // a list
        +Actor,                    // an atom
        +Action,                   // a ground literal
        -ActorVpExpls,             // a list of lists
        -ObsVpExpls,               // a list of lists
        -ActLit,                   // a literal
        -ObsLit                    // a literal
    );
    ...
```

Listing 1: Usage of the tomabd.agent.tom_abduction_task IA. A "+" precedes variables that must be bound at invocation time, while a "-" precedes variables that are bound by the IA.

Chapter 4] and the customisation of Jason components [7, Chapter 7]. Our implementation is documented and publicly available under a Creative Commons license.[5] It has been packaged into a Java Archive (.jar) file to facilitate its use as an external library for developers who want to include it in their applications.

The core of the implementation consists of the tomabd.agent.TomAbdAgent class, a subclass of Jason's default agent class. It contains all the methods to implement the functions in Figure 3 (plus some auxiliaries). Besides the main BB (inherited from Jason's default agent class), tomabd.agent.TomAbdAgent also has a backup BB. The BUF is part of Jason's default agent class, which we override in our implementation to include a call to EUF after percepts have been updated. The abductive reasoner implementing the ABDUCE function is included as a set of Prolog-like rules in an AgentSpeak file, which the agent class automatically includes at initialization time.

A call to the main function of this agent model (TOMABDUCTIONTASK) is not included within Jason's native reasoning cycle, and hence does not constrain it in any way. Instead, an internal action (IA), tomabd.agent.tom_abduction_task, is provided as an interface to the agent's method. The usage of this IA is illustrated in Listing 1. Calling TOMABDUCTIONTASK through an IA allows the agent developer flexibility and control over when to trigger it from within the application-specific agent code. Furthermore, the invocation of TOMABDUCTIONTASK from an IA ensures that the whole function is executed within one act() step of the BDI reasoning cycle. Therefore, its execution does not interfere with changes in the BB that happen during other perceive()

[5] https://github.com/nmontesg/tomabd

```
+!trigger : context
    <- ... ;
    tomabd.agent.select_action(Action);
    Action.                                  // execute action on the environment

-!trigger[error(ia_failed),
    error_msg("internal action tomabd.agent.select_action failed")]
    <- ?default_action(DefAct);              // look for a default action in the BB
    DefAct.
```

Listing 2: Usage of the tomabd.agent.select_action IA and a possible contingency plan to handle its failure.

or act() steps (e.g. belief removal or addition operations) of the BDI reasoning cycle.

We have exposed the reasons why TomAbductionTask is not called from within the agent's reasoning cycle, but using an IA interface. In summary, through an IA the TomAbd agent model provides additional functionalities to Jason agents, without restricting the use of other custom components nor placing constraints on the BDI reasoning cycle. Similar remarks apply to the SelectAction function and its counterpart IA tomabd.agent.select_action (also included in our implementation), which operates similarly to tomabd.agent.tom_abduction_task but provides an interface to SelectAction instead. It is called as tomabd.agent.select_action(A), where A is a free variable bounded by the IA to the return value of SelectAction.

As covered in Section 3.4, if using the default implementation of SelectAction (or any other implementation that may return *null*), contingency measures should be put into place to handle the possibility of failure. In Listing 2 we propose a strategy to do this. In the first plan, the agent uses the tomabd.agent.select_action IA to decide which action to perform next. If the IA is successful, the agent goes on to execute it as a standard action on the environment. If not, the second plan in Listing 2 handles the failure. The annotations in this plan, namely the error and error_msg literals, ensure that this plan handles only the failure of tomabd.agent.select_action, not of any other source of failure in the previous plan (e.g. the failure of execution of Action on the environment). In Listing 2, if tomabd.agent.select_action fails, the agent queries its BB to look for a default action, and executes it on the environment.

In Listings 1 and 2, the IAs tomabd.agent.tom_abduction_task and tomabd.agent.select_action are invoked as part of the body of agent plans. Nonetheless, similarly to standard Jason IAs, they may also appear in the context of plans. If that is the case, the execution of the corresponding TomAbductionTask and SelectAction would be moved to the deliberate() step of the BDI reasoning cycle. Whether it is more desirable to have the mentioned functions execute

```
public class MyAgent extends TomAbdAgent {

    @Override
    public ListTermImpl erf(ListTermImpl expls) {
        Literal obsViewpoint = Literal.parseLiteral("viewpoint(observer)");
        Literal actViewpoint = Literal.parseLiteral("viewpoint(actor)");
        Unifier un = new Unifier();
        if (believes(actViewpoint, un)) {
            return erfAct(expls);
        } else if (believes(obsViewpoint, un)) {
            return erfObs(expls);
        } else {
            throw new RuntimeException("Error in erf: no valid viewpoint belief");
        }
    }

}
```

Listing 3: Customisation of the ERF function, based on whether the agent is currently adopting the actor or the observer's viewpoint.

in an act() step (by placing their corresponding IAs in the plan body) or in a deliberate() step (by placing them in the plan context) is a decision for the agent developer to take.

In summary, of the agent functions displayed in Figure 3, only TomAbductionTask and SelectAction have a correspondinging IA interface, with SelectAction being the only one of the two that is customisable. Additionally, the ERF and the EUF are also customisable, but these are called from within other functions and hence are not accompanied by an IA interface.

To override the default implementation of any of these functions, the developer needs to write new erf(), euf() and selectAction() methods in an agent subclass of tomabd.agent.TomAbdAgent. For example, Listing 3 provides an agent subclass with an alternative implementation of ERF that applies a different revision function depending on whether the agent is currently working at the observer's or at the actor's perspective.

5 Results

5.1 Experimental Setting

As a proof of concept, we have applied the TomAbd agent model to the Hanabi domain presented in Section 2.3, for teams of 2 to 5 players in self-play

mode.[6] This means that the teams are homogeneous, composed exclusively of TomAbd agents. As for the `action/2` clauses that implement the team strategy, Hanabi has a thriving community of online players that have gathered a set of *conventions* for the game, called the H-group conventions.[7] These conventions comprise definitions (e.g. what constitutes a *save hint* or a *play hint*) and guidelines to follow during game play. We have taken inspiration from these conventions to devise our action selection clauses. However, while these conventions are itemised according to player experience, we have only made use of the introductory-level ones. Our goal is not to synthesise the playing strategy that achieves the maximum possible score, but to explore the usefulness of the capabilities of the TomAbd model in an example domain. We leave the exploration of more sophisticated conventions for future work.

To trigger the execution of the TomAbductionTask function, participants publicly broadcast their action of choice prior to execution. To handle these announcements, we define a Knowledge Query and Manipulation Language (KQML) custom performative, publicAction. Agents react to messages with this performative by executing the tomabd.agent.tom_abduction_task IA using first-order ToM. This means that, when adopting the other acting agent's viewpoint, agents do not take that perspective through any intermediate agents. Hence, agents work with program $T_{i,l}$, where i is the observer and l is the acting agent, when adopting the actor's viewpoint to generate explanations. Consequently, the variable ObsVp in Algorithm 2 is bound to the empty list "[]". Additionally, all the generated literals from the abductive explanations are immediately incorporated into the agent's program.

We evaluate the performance of the TomAbd agent model for the Hanabi domain, using the basic set of H-group conventions and first-order ToM. We ran 500 games with random seed 0 to 499, for every team size and switching on/off the call to TomAbductionTask. The simulations were distributed over 10 nodes at the high performance computing cluster at IIIA-CSIC.[8]

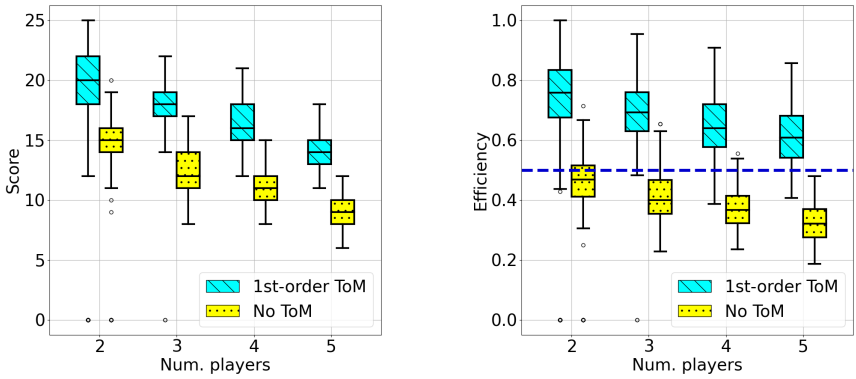
5.2 Score and Efficiency

The results are first evaluated in terms of the absolute score at the end of every game. This is the most straightforward performance metric and one that allows comparison with other work on Hanabi AI. Beyond the absolute score, we also evaluate teams according to their *communication efficiency*, which we define as the ratio between the final score and the total number of hints given during the course of a complete game. This metric quantifies how efficient the team is at turning communication (i.e. hints) into utility (i.e. score). Intuitively, a lower bound for the efficiency metric is $\frac{1}{2}$, as two hints are needed (one for colour and one for rank) to completely learn about a card's identity and be able to safely play it.

[6] The code that applies the TomAbd model to Hanabi is available at https://github.com/nmontesg/tomabd/examples/hanabi

[7] https://hanabi.github.io/

[8] https://www.iiia.csic.es/en-us/research/ars-magna/



(a) (b)

Fig. 4: Results for the score (a) and communication efficiency (b). Cyan ruled boxes correspond to games where agents make use of the capabilities of the TomAbd agent model, and yellow dotted boxes correspond to games where they do not. The dashed line on the efficiency plot indicates the bound of two hints per score point.

Box plots for the results of performance in terms of score and efficiency are displayed in Figure 4. Additionally, the experimental distributions are available in Figure 8 in the Supplementary information. Visually, Figure 4 conveys that the incorporation of ToM and abductive reasoning capabilities boosts performance, both in terms of score and efficiency. Furthermore, regardless of team size, the efficiency is over the lower bound for over 75% of the games when the ToM and abduction capabilities are used. In contrast, when these cognitive abilities are switched off, the efficiency falls below the lower bound for approximately 75% of the runs.

In Table 1 the means and standard deviations for the score and communication efficiency are provided. Moreover, the average percentage increase (comparing pairs of games with the same random seed with and without calls to TomAbductionTask) is displayed in the *Improvement* row for every team size. The average scores in Table 1, even with the ToM and abductive reasoning switched on, are still far from the current state-of-the-art in Hanabi AI, with average score of up to 24.6 [49]. Nonetheless, they are in line with the performance of current rule-based Hanabi-playing bots (see Table 1 by Siu et al. [42]). Moreover, recall that the goal of this work is not to synthesise the optimal team strategy for Hanabi, but to develop a domain-independent agent model capable of putting itself in the shoes of other agents and reasoning from their perspective. The Hanabi game was selected as a test bed for this model, alongside a very simplistic playing strategy. Yet, we anticipate that the results

Table 1: Average, standard deviation and improvement when using the TomAbductionTask function for the score and communication efficiency. Paired samples t-test confirmed that the average score and efficiency are significantly better when the TomAbductionTask function is used, regardless of team size.

Num. players	TomAbductionTask	Score	Efficiency
2	Yes	18.61 ± 5.92	0.71 ± 0.22
	No	14.57 ± 2.93	0.46 ± 0.10
	Improvement	27%	54%
3	Yes	17.97 ± 1.94	0.70 ± 0.10
	No	12.52 ± 1.56	0.42 ± 0.07
	Improvement	45%	71%
4	Yes	16.50 ± 1.61	0.64 ± 0.09
	No	11.23 ± 1.36	0.38 ± 0.06
	Improvement	49%	75%
5	Yes	14.42 ± 1.37	0.62 ± 0.09
	No	9.23 ± 1.30	0.33 ± 0.06
	Improvement	59%	91%

presented here could be improved through the introduction of more advanced playing conventions, such as "prompts" and "finesses".

To confirm the observation that performance is better when agents make use of the TomAbductionTask function, we used statistical testing. First, we applied the Shapiro-Wilk test of normality [53] to test that the score and efficiency distributions in Figure 8 are normally distributed, under all the experimental conditions. We confirm that this is indeed the case for confidence level 99%. Second, we used the paired samples t-test [54] to confirm that the averages for the score and the efficiency, across all team sizes, are significantly better when the TomAbduction function is used. We used the paired samples version of the t-test, rather than the independent samples, because games with equal random seeds are related as far as the sequence of cards that are dealt from the deck is the same for all. The results confirm that the averages for the score and the efficiency are significantly better when the TomAbductionTask function is called with respect to when it is not, across all team sizes and for confidence level 99%. Therefore, we conclude that the use of the TomAbductionTask function quantitatively boosts the performance of teams, independently of their size.

Once we confirmed that, indeed, the execution of the TomAbductionTask function produces significantly better performance in terms of score and efficiency, we sought to quantify this improvement. As explained earlier, games of equal team size and random seed are related since the sequence of dealt cards is the same for both. For this reason, it makes sense to compare

the score and the efficiency for games with the ToM capabilities on and off, while controlling for team size and seed. To do this, we computed the percentage increase in the score and efficiency when using the TomAbductionTask function, and then aggregated these values into the average across all random seed. These results are displayed in the *Improvement* row in Table 1. They show that there is indeed a notable percentage increase in both score and efficiency, and this improvement increases monotonically with team size. For example, the increase in score is around 30% for teams of two players while it reaches almost 60% for the largest teams (five players).

5.3 Elapsed Time

The results presented in the previous section clearly prove that the use of the TomAbductionTask function (using first-order ToM and with the selected action selection rules) has a positive effect on the team performance, both in their final score and the efficiency of communication. In this section, we analyse the computational load associated to this performance boost.

In Figure 5 we present the results for the elapsed time of the TomAbductionTask function. Every box contains data on at least 4,000 runs of the function. The samples in Figure 5 correspond to the execution of the TomAbductionTask function across different games with different random seeds, and at different stages of the game.

In all cases, the execution of the function has magnitude in the hundreds of milliseconds. As expected, the elapsed time tends to increase and fall within a larger range as the team size increases. This is due to the larger BB that agents have to manage when they are part of a larger team. This results in a larger space to search through in order to construct the abductive explanations. For example, for teams of size 2, agents have 10 percepts concerning the rank and

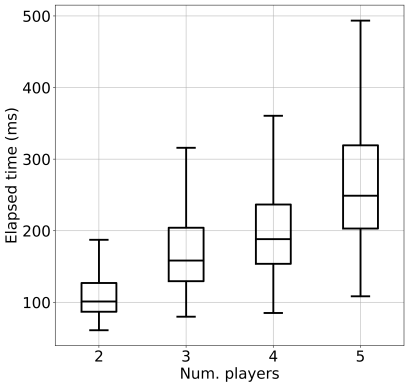


Fig. 5: Execution time of the TomAbductionTask function for the Hanabi domain with different team sizes.

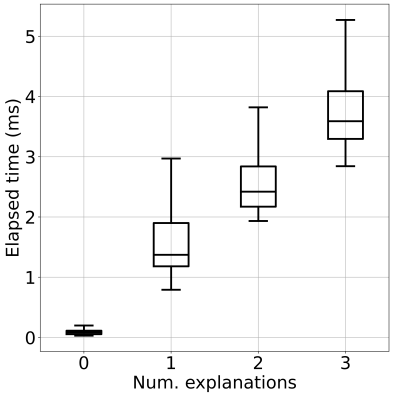


Fig. 6: Execution time of the EUF for the Hanabi domain, as a function of the abductive explanations in the agent's BB at execution time.

colour of the cards of their fellow player. Meanwhile, for teams of size 5, agents have 32 percepts about the cards of others.

As explained in Section 4, the TomAbductionTask function (and also SelectAction) is executed through an IA at the discretion of the developer. Hence, it is not natively integrated into the BDI reasoning cycle. Nonetheless, there is one TomAbd-specific function that *is* called from the BDI reasoning cycle: EUF, which is called from BUF, a central component of the sensing step in the Jason reasoning cycle. Hence, to quantify the burden put on the BDI reasoning cycle by the TomAbd agent model, we have to analyse the performance of EUF.

In Figure 6 we present the results for the elapsed time of EUF. Every box contains at least 750 data points. The results are itemized by the number of explanations in the agent's BB at the time EUF was executed, since our default implementation of EUF loops over the literals in the BB that originated from an abductive reasoning process. There were no instances found with 4 or more explanations. The results in Figure 6 show that, for the first order ToM we are using for the Hanabi domain, EUF entails a negligible overhead on the execution time of the Jason reasoning cycle. Its execution time is around two orders of magnitude smaller than that of TomAbductionTask and, as expected, follows an approximately linear trend with respect to the number of abductive explanations in the BB. Nonetheless, we expect the execution time of EUF to increase as higher-order ToM is introduced.

5.4 Information Gain

The previous analyses quantify the overall outcome of a game, either in terms of score or efficiency, and their computational requirements. Now, in the current and the following section, we would like to quantify the amount and

the value of the information that agents derive from the execution of the TomAbductionTask function.

The analysis that follows relies on some features that are specific to Hanabi and hence not generally exportable to other domains where the TomAbd agent model may be applied. The first enabling feature is the fact that Hanabi has a well-defined set of states that the game might be in at any given moment. These states are defined by the heights of the stacks, the available information tokens, the number of lives remaining and the cards in the discard pile, which are all observable by all players. Additionally, states are also characterised by the cards at each player's hand, which are not common knowledge.[9]

In game theoretical terms, the above feature is referred to as Hanabi being a game of imperfect yet complete information. In other words, players in Hanabi do not in general have access to all the information characterising the current state of the game, but they can infer a finite set of states the game might be in. Additionally, using domain knowledge (namely, the number of duplicate identical cards, which depends on their rank) and, potentially, the abductive explanations currently in their BB, agents can compute, for every slot S in their hand, the marginal probability distribution for the colour and the rank of their card in S. By examining these probability distributions and comparing them to the true one (which assigns unit probability to the colour and rank of the actual card a player holds in S, and zero otherwise), we can quantify the information gain, which we present in the present section.

The second feature of Hanabi that enables the analysis on information value in Section 5.5 is the fact that, as any classical game, Hanabi has a set of well-defined end-states with an assigned numerical utility or score. This

[9]The sequence of cards in the deck, which is hidden to all players, might also be considered as part of the state description in Hanabi. However, we prefer to view it as a randomising device rather than as part of the state description. In any case, its treatment is not relevant to the analysis in Sections 5.4 and 5.5.

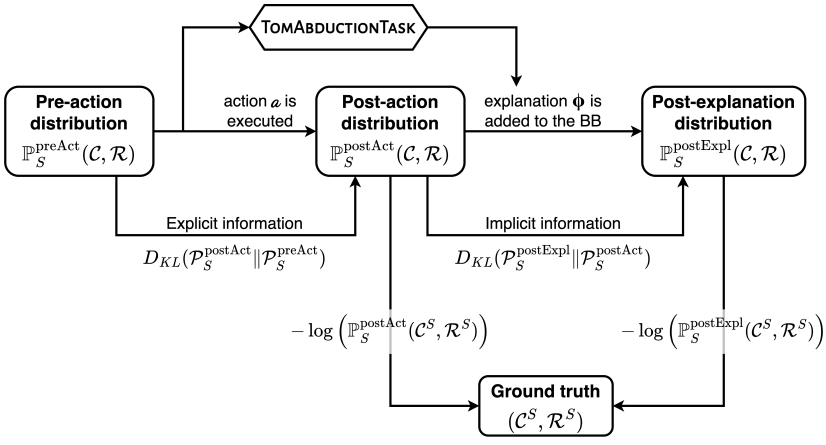


Fig. 7: Outline of the probabilistic analysis of the simulation results.

characteristic, together with the previous one, allows us to relate the reduction in uncertainty of the probability distributions over the cards in player's slots with the increase in score when the ToM and abductive reasoning capabilities of the TomAbd agent model are introduced.

We begin, then, by quantifying the gain in information derived from the combination of ToM and abductive reasoning. To help with this, consider Figure 7. Agents maintain a marginal probability distribution over the identity of the card at each of their slots, i.e. the tuple $(\mathcal{C}, \mathcal{R})$ of random variables corresponding to the card's colour and rank. At every turn of the game, there are three distributions to consider: the pre-action distribution *before* the action is executed $\mathbb{P}_S^{\text{preAct}}$, the post-action distribution *after* the action is executed $\mathbb{P}_S^{\text{postAct}}$, and the post-explanation distribution after the action is executed *and* the abductive literals derived from the TomAbductionTask function have been introduced into the agent's BB $\mathbb{P}_S^{\text{postExpl}}$. In addition to these three distributions, the true identity of a card at slot S is denoted as $(\mathcal{C}^S, \mathcal{R}^S)$. We refer to $(\mathcal{C}^S, \mathcal{R}^S)$ as the *ground truth* for slot S. Trivially, the true probability distribution can be considered to be $\mathbb{P}_S^*(\mathcal{C}^S, \mathcal{R}^S) = 1$ and 0 otherwise.

To quantify the distance between two probability distributions, we use the Kullback-Leibler divergence [55], which defines the relative entropy from distribution Q to distribution P as:

$$D_{KL}(P\|Q) = \sum_{x_i \in \mathcal{X}} P(x_i) \log \left(\frac{P(x_i)}{Q(x_i)} \right) \tag{6}$$

If $P(x_i) = 0$ for some i, the contribution of the i-th term is assumed to be null.

The Kullback-Leibler distance quantifies how much information is lost when approximating P using Q or, alternatively, how much information is gained by refining Q into P. In the Hanabi game, we are working with probability distributions over the domain of card identities, which has size 25 (5 colours $\times$ 5 ranks). Therefore, for all our computations we take the logarithm in Equation (6) with base 25.

We evaluate the gain provided by the abductive explanations by comparing the distance to the ground truth between the post-action and the post-explanation distributions at every game turn. Since the ground truth corresponds to a single card identity with probability 1, the Kullback-Leibler distance from the two aforementioned distributions to the ground truth is reduced to:

$$D_{KL}(\mathbb{P}_S^* \| \mathbb{P}_S^{\{\cdot\}}) = -\log \left(\mathbb{P}_S^{\{\cdot\}}(\mathcal{C}^S, \mathcal{R}^S) \right) \tag{7}$$

The results for the percentage reduction in distance between the post-action and the post-explanation distribution to the ground truth appear in Table 2. The reduction in distance is large across all teams sizes, starting at around 85% for teams of 2 players, and increasing monotonically with team size up to a 91% for teams of 5 players.

Table 2: Reduction is distance to the ground truth from the post-action to the post-explanation distribution.

Num. players	%
2	85.33
3	88.29
4	89.43
5	91.49

5.5 Information Value

The previous results indicate that the incorporation of abductive explanations does shrink the distance to the ground truth to a very large extent. However, the analysis does not indicate how *valuable* the information derived from these abductive explanations is. In other words, how much score are agents able to draw from the information provided by abductive explanations.

To quantify the score value of abductive explanations, we define the two following quantities (see Figure 7). First, we define the *explicit information gain* as the Kullback-Leibler distance from the pre-action distribution to the post-action distribution. Second, we define the *implicit information gain* as the Kullback-Leibler distance from the post-action distribution to the post-explanation distribution. The explicit information gain quantifies the knowledge acquired just from observing the progress of the game, as new cards are drawn and revealed. Meanwhile, the implicit information gain quantifies the knowledge derived only from the abductive explanations.

Next, we define the *total explicit information gain* (TEIG) as the sum across all slots S and moves m_i over the course of a game of the explicit information gain:

$$\text{TEIG} = \sum_{m_i} \sum_{S} D_{KL}(\mathbb{P}_S^{\text{postAct}} \| \mathbb{P}_S^{\text{preAct}}) \tag{8}$$

The *total implicit information gain* (TIIG) is defined analogously to Equation (8), but using the distance from the post-action to the post-explanation distribution, $D_{KL}(\mathbb{P}_S^{\text{postExpl}} \| \mathbb{P}_S^{\text{postAct}})$, instead. The TEIG is defined for all games, regardless of whether agents are using the TomAbductionTask function. The TIIG is defined only for games where the mentioned function is active. For these games, we compute the percentage of information that is derived from the ToM and abduction capabilities as:

$$\%\ \text{implicit info.} = \frac{\text{TIIG}}{\text{TIIG} + \text{TEIG}} \cdot 100 \tag{9}$$

Then, to quantify the contribution of each type of information to score, we start by computing the *explicit score rate* (ESR) as the ratio of the score to the TEIG, for games where agents are *not* using the TomAbductionTask function. Once we have the ESR, we turn to games where agents *are* using this

Table 3: Average percentage of implicit information and average score assigned to this implicit information.

Num. players	% implicit info.	% implicit score
2	15.15	27.80
3	15.51	30.55
4	18.79	32.24
5	19.40	38.37

function, and we estimate the residual score that cannot be explained away by the explicit information that agents acquire by observing the evolution of the system as:

$$\text{residual score} = \text{score} - \text{ESR}_{\text{seed}} \cdot \text{TEIG} \tag{10}$$

where ESR_{seed} is the ESR for the game without calls to TomAbductionTask with the same random seed, and TEIG (TIIG) is the total explicit (implicit) information gain for the game that employs the TomAbductionTask function. We use the ratio between the residual score in Equation (10) and the total score as the estimation of the contribution of the implicit information to the overall performance of the team.

The results for the average percentage of implicit information and the average percentage score that can be assigned to this explicit information appear in Table 3, for games where agents use the TomAbductionTask function. Across all team sizes, the information derived from the ToM and abduction capabilities accounts for between 15% and 20% of the total information. However, this implicit information accounts for disproportionate amount of the final score, between 27% and 40% of it. Therefore, when agents use the capabilities of the TomAbd model, the information derived from these capabilities ends up being overrepresented in the final score by a factor of between $\times 1.7$ and $\times 2.0$.

6 Related Work

This section compares our contribution with related approaches. Previous work on Theory of Mind implementations in agent-oriented programming have, for the most part, used languages based on the BDI architecture [29, 30, 56, 57]. This is a natural choice that we share, since BDI-based languages provide constructs for the mental states that ToM estimates and operates on. Specifically, some ToM implementations are, like our TomAbd agent model, developed in Jason [29, 30, 56]. In contrast, other work uses Extended 2APL [57].

Panisson et al. [30] implement ToM for deceptive purposes. They focus on the communicative interventions, i.e. the requesting and sharing of (possibly untruthful) information, and provide operational semantics [56] for the effects that these actions have of the models that agents maintain of one another. Their approach is very much in line with the TT account of ToM. It uses dedicated predicates to infer additional mental states, such as goals and future actions, given prior beliefs. These inferences are made from within the agent

program, a feature which we consider qualifies as adherence to the theoretical version of ToM.

Sarkadi et al. [29] extend the previous model by incorporating elements of trust and modelling several agent profiles based on their attitudes. In this extension, they distinguish between TT and ST components within their model. They argue that the TT component handles the assignment of prior beliefs to other agents, while the ST component handles inferences based on those. In our work, we do not distinguish between TT and ST components, but consider that our approach overall aligns more closely with the ST account of ToM than with the TT account.

Harbers et al. [57] establish a different criterion for classifying ToM approaches into TT and ST. They develop two separate ToM implementations, one identified with TT and the other with ST, for applications in virtual training systems. Both architectures maintain knowledge bases for the beliefs, logical rules and goals of other agents. The difference between the ST and TT approaches is found in the reasoner that is applied to the knowledge bases assigned to other agents. The TT architecture applies rules about how other agents combine their beliefs, goals and plans, which are explicitly included as part of the agent's own knowledge. In contrast, the ST architecture uses the agent's native reasoner, making it more lightweight. Besides this, other advantages were found for the ST architecture with respect to the TT one, namely code reusability and flexibility to deal with non-BDI agents.

The choice to maintain belief bases for other agents, in addition to the agent's own belief base, is very different to the TomAbd agent model, where we generate estimations of the beliefs of others on demand at run-time, using the set of ToM rules as a meta-interpreter. This allows the TomAbd model to engage in higher-order ToM by recursively applying the set of ToM rules. In comparison, the maintenance of belief bases for other agents hinders the use of ToM beyond first-order. For every recursive path that the agent would like to take into account, i.e. what we refer to as the *viewpoint* in Definition 3, a different knowledge base would have to be initialised and updated throughout the agent's lifetime, resulting in a rapid combinatorial explosion in memory requirements. This limitation to first-order ToM is also shared by other work [29, 30].

There is an important difference in the focus of ToM between the works reviewed in this section and the TomAbd agent model of this paper. In related work [29, 30, 56, 57], the purpose of the ToM functionalities is to compute the action that best pursues the agent's goal, whether it is to deceive an opponent or to provide explanation to assist in staff training. Hence, ToM is directed towards the *deliberation* step of the BDI reasoning cycle. In contrast, in our approach, ToM is directed towards the *sensing* step, with the TomAbd model computationally implementing the cognitive processes to use other agents as sensors. Accordingly, the core function of the TomAbd agent model is TomAbductionTask, which uses abduction to compute explanations either about the state of the environment or the mental state of other

agents. The execution of this function results in the agent being in a more informed position when it comes to its own decision-making.

It should be noted that, even if at this current stage the TomAbd agent model strongly links ToM with sensing, it provides an avenue to include these capabilities into the agent's deliberation stage too. The component directed towards practical reasoning, the SelectAction function, has not thus far received as much attention as TomAbductionTask. Nonetheless, as mentioned previously, this function is a *customisable* component of the model. This leaves a lot of room to develop further implementations that more explicitly use the ToM capabilities of the agent during the deliberation stage, for example by making calls to the AdoptPerspective procedure within SelectAction.

7 Conclusions

In this paper, we have presented the novel TomAbd model, an agent architecture combining Theory of Mind and abductive reasoning. Its main functionality is the ability to perceive the state of the system through the eyes of their peers, and infer the beliefs that account for their most recent action using abductive reasoning. This core functionality is accompanied by other functions that handle how the abductive explanations are refined, updated and used during practical reasoning.

There are four features that make the TomAbd agent model stand out. First, the model is able to handle ToM of an arbitrary order without additional memory requirements. Second, our approach has a strong preference for a simulation account of ToM over a theory account. Third, we emphasise the role of ToM for sensing over deliberation. The goal of ToM in our model is to extract the information as perceived by other agents, hence using them as proxies for obtaining data about the world. Finally, we would like to highlight the user-friendliness and the flexibility of our implementation, which allows customisation of many of its components.

We have tested our model in the benchmark domain of Hanabi. Our results show that teams whose agents use ToM consistently perform better than those that do not, both in terms of absolute score and efficiency of communication. In terms of information gain, our analysis shows that the knowledge derived from the abductive reasoning component of the model greatly reduces uncertainty. Additionally, the information derived from the combination of ToM and abductive reasoning contributes to the final score in a disproportionate amount, with respect to the explicit information derived from the observation of the evolution of the game alone.

The TomAbd agent model presented here offers several directions for future work. First, within the Hanabi game domain, an option would be to investigate more sophisticated action selection rules. Additionally, it would be interesting to investigate the perception that human players have of TomAbd teammates, for strategies of different skill levels. This research could shine

light on how well is human ToM captured by the agents, and how compatible is human ToM and the artificial ToM we have presented here.

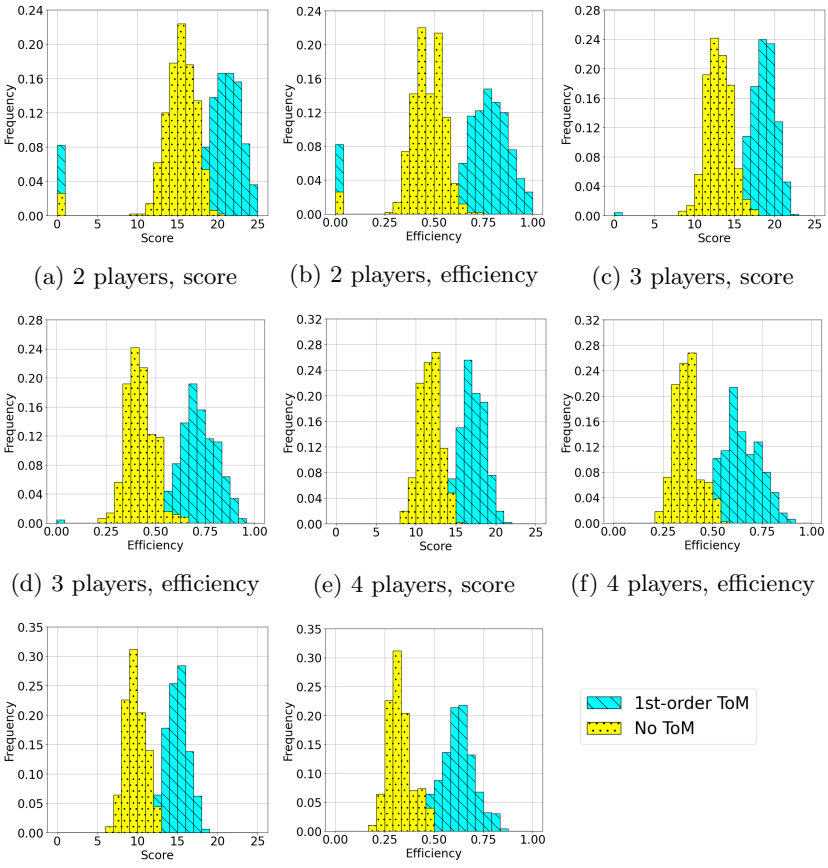
Second, the TomAbd agent model can be applied to other domains where ToM capabilities may entail a potential benefit, with the goal of extracting the common general features that a domain must have in order for ToM to result in improved performance. This research could also expand the set of customised functions for explanation revision and update, as well as the incorporation of ToM in the deliberation stage. Furthermore, the application to other domains would require the development of additional metrics to quantify the benefits entailed by the agents' ToM capabilities, analogous to the information gain and information value metrics we present in this paper for Hanabi. Such metrics would naturally need to consider the domain properties such as whether there is a closed set of states and/or any heuristics available to quantify the value of MAS states.

Third, the flexibility of the TomAbd model could be enhanced by extending the type of constructs for which the TomAbd agent model is able to provide explanations. In other words, with small additional functionality, TomAbd agents could be adapted to compute abductive explanations not just for actions, but for mental states such as beliefs, goals and intentions. Of course, the mental state that is taken as input to the machinery of the TomAbd agent model must either be the result of some observation (e.g. agent i overhears agent j discuss its goals with a third party), or of other techniques, such as goal recognition, that aggregate granular observations into a mental state, i.e. a sequence of atomic actions into the goal or intention pursued by those actions.

Regardless of the modality of the observation, the process of generating an explanation for it would be analogous to that presented in the TomAbd agent model for actions. In summary, as long as some agent i has an input about another agent j (such as an action j has taken, a belief or a desire j holds, or an intention j is pursuing) and an estimation of the inference rules that j is using, i can provide an explanation for the input. Of course, its precision will depend on the accuracy of the input and of the inference rules that i believes j to have.

Last but not least, the computational requirements versus the performance benefits of using higher-order Theory of Mind, in the Hanabi game or in other domains, presents an interesting challenge. Here too, many questions arise. For example, does performance plateau around a particular recursion level n_{pl}? Is n_{pl} a domain-independent quantity? How does it compare with respect to the maximum order of ToM that humans are able to cope with? Does this have any evolutionary implications? In other words, did humans develop ToM just far enough to obtain the maximum evolutionary advantage, but not any further to save resources?

To conclude, our work presents and tests a novel model for agents with Theory of Mind. It provides the cognitive machinery to adopt the perspective of a peer and reason from its perspective. It is inspired by the though processes



(a) 2 players, score (b) 2 players, efficiency (c) 3 players, score

(d) 3 players, efficiency (e) 4 players, score (f) 4 players, efficiency

(g) 5 players, score (h) 5 players, efficiency

Fig. 8: Experimental distributions of the absolute score and the communication efficiency. Cyan ruled boxes correspond to games where agents make use of the capabilities of the TomAbd agent model, and yellow dotted boxes correspond to games where they do not. The Shapiro-Wilk test confirmed that all experimental histograms follow a normal distribution.

that humans engage in when trying to understand the motivations for the behaviour of others. Our model endows autonomous agents with essential social abilities, that are becoming increasingly important in the current AI landscape.

Supplementary information.

Acknowledgments. N. Montes, N. Osman and C. Sierra would like to thank the TAILOR connectivity fund for funding their research visit to KCL.

Declarations

Competing interests. The authors declare that they have no competing interests that could have influence this work.

Authors' contributions. N. Montes wrote the main manuscript text. All authors reviewed the manuscript.

Funding. N. Montes, N. Osmain and C. Sierra acknowledge funding from the Spanish funded VAE project (#TED2021-131295B-C31), the EU VALAWAI project (HORIZON #101070930), and the EU TAILOR project (H2020 #952215).

Availability of data and materials. All the code accompanying this work can be freely accessed here. All the simulation data generated is freely available here.

References

[1] Dafoe, A., Bachrach, Y., Hadfield, G., Horvitz, E., Larson, K., Graepel, T.: Cooperative AI: machines must learn to find common ground. Nature **593**(7857), 33–36 (2021). https://doi.org/10.1038/d41586-021-01170-0

[2] Paiva, A., et al.: WP6 – Social AI: Learning and Reasoning in Social Contexts. https://www.tailor-social-ai.eu/home. ICT-48 TAILOR: Foundations of Trustworthy AI – Integrating Reasoning, Learning and Optimization (2020)

[3] Malle, B.: In: Biswas-Diener, R., Diener, E. (eds.) Theory of Mind. DEF publishers, Champagne, IL (2022)

[4] Knobe, J.: Theory of mind and moral cognition: exploring the connections. Trends in Cognitive Sciences **9**(8), 357–359 (2005). https://doi.org/10.1016/j.tics.2005.06.011

[5] Williams, J., Fiore, S.M., Jentsch, F.: Supporting artificial social intelligence with theory of mind. Frontiers in Artificial Intelligence **5** (2022). https://doi.org/10.3389/frai.2022.750763

[6] Montes, N., Osman, N., Sierra, C.: Combining theory of mind and abduction for cooperation under imperfect information (2022) https://arxiv.org/abs/2209.15279 [cs.MA]

[7] Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak Using Jason. John Wiley & Sons, New York, NY, USA (2007)

[8] Frith, C., Frith, U.: Theory of mind. Current Biology **15**(17), 644–645 (2005). https://doi.org/10.1016/j.cub.2005.08.041

[9] Korkiakangas, T., Dindar, K., Laitila, A., Kärnä, E.: The Sally–Anne test: an interactional analysis of a dyadic assessment. International Journal of Language & Communication Disorders **51**(6), 685–702 (2016). https://doi.org/10.1111/1460-6984.12240

[10] Baron-Cohen, S., Leslie, A.M., Frith, U.: Does the autistic child have a "theory of mind" ? Cognition **21**(1), 37–46 (1985). https://doi.org/10.1016/0010-0277(85)90022-8

[11] Tager-Flusberg, H.: Evaluating the theory-of-mind hypothesis of autism. Current Directions in Psychological Science **16**(6), 311–315 (2007). https://doi.org/10.1111/j.1467-8721.2007.00527.x

[12] Askham, A.V.: 'Theory of mind' in autism: A research field reborn. Spectrum (2022). https://doi.org/10.53053/gxnc7576

[13] Röska-Hardy, L.: Theory theory (simulation theory, theory of mind). In: Encyclopedia of Neuroscience, pp. 4064–4067. Springer, Berlin Heidelberg (2008). https://doi.org/10.1007/978-3-540-29678-2_5984

[14] van der Hoek, W.: Systems for knowledge and belief. Journal of Logic and Computation **3**(2), 173–195 (1993). https://doi.org/10.1093/logcom/3.2.173

[15] Rendsvig, R., Symons, J.: Epistemic Logic. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy, Summer 2021 edn. Metaphysics Research Lab, Stanford University, Stanford, CA (2021)

[16] Meyer, J.-J.C., Broersen, J., Herzig, A.: BDI logics. In: van Ditmarsch, H., Halpern, J.Y., van der Hoek, W. (eds.) Handbook of Epistemic Logics. College Publications, Rickmansworth, WD3 1DE (2015). Chap. 10

[17] Corballis, M.: The uniqueness of human recursive thinking. American Scientist **95**(3), 240 (2007). https://doi.org/10.1511/2007.65.240

[18] Corballis, M.C.: The Recursive Mind: The Origins of Human Language, Thought, and Civilization, p. 291. Princeton University Press, Princeton, NJ (2011)

[19] Albrecht, S.V., Stone, P.: Autonomous agents modelling other agents: A comprehensive survey and open problems. Artificial Intelligence **258**, 66–95 (2018). https://doi.org/10.1016/j.artint.2018.01.002

[20] Baarslag, T., Hendrikx, M.J.C., Hindriks, K.V., Jonker, C.M.: Learning about the opponent in automated bilateral negotiation: a comprehensive survey of opponent modeling techniques. Autonomous Agents and Multi-Agent Systems **30**(5), 849–898 (2015). https://doi.org/10.1007/

s10458-015-9309-1

[21] Nashed, S., Zilberstein, S.: A survey of opponent modeling in adversarial domains. Journal of Artificial Intelligence Research **73**, 277–327 (2022). https://doi.org/10.1613/jair.1.12889

[22] Rabinowitz, N., Perbet, F., Song, F., Zhang, C., Eslami, S.M.A., Botvinick, M.: Machine theory of mind. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 4218–4227. PMLR, Stockholm, Sweden (2018). https://proceedings.mlr.press/v80/rabinowitz18a.html

[23] Wang, Y., Zhong, F., Xu, J., Wang, Y.: Tom2c: Target-oriented multi-agent communication and cooperation with theory of mind. In: International Conference on Learning Representations (2022). https://openreview.net/forum?id=M3tw78MH1Bk

[24] Jara-Ettinger, J.: Theory of mind as inverse reinforcement learning. Current Opinion in Behavioral Sciences **29**, 105–110 (2019). https://doi.org/10.1016/j.cobeha.2019.04.010. Artificial Intelligence

[25] Cuzzolin, F., Morelli, A., Cîrstea, B., Sahakian, B.J.: Knowing me, knowing you: theory of mind in AI. Psychological Medicine **50**(7), 1057–1061 (2020). https://doi.org/10.1017/s0033291720000835

[26] de Weerd, H., Verheij, B.: The advantage of higher-order theory of mind in the game of limited bidding. In: Workshop on Reasoning About Other Minds: Logical and Cognitive Perspectives, vol. 751, pp. 149–164 (2011)

[27] de Weerd, H., Verbrugge, R., Verheij, B.: Higher-order social cognition in the game of rock-paper-scissors: A simulation study. In: Bonanno, G., Van Ditmarsch, H., Hoek, W. (eds.) Proceedings of the 10th Conference on Logic and the Foundations of Game and Decision Theory (LOFT 2012), pp. 218–232 (2012)

[28] de Weerd, H., Verbrugge, R., Verheij, B.: Higher-order theory of mind is especially useful in unpredictable negotiations. Autonomous Agents and Multi-Agent Systems **36**(2) (2022). https://doi.org/10.1007/s10458-022-09558-6

[29] Ştefan Sarkadi, Panisson, A.R., Bordini, R.H., McBurney, P., Parsons, S., Chapman, M.: Modelling deception using theory of mind in multi-agent systems. AI Communications **32**, 287–302 (2019). https://doi.org/10.3233/AIC-190615

[30] Panisson, A., Mcburney, P., Parsons, S., Bordini, R., Sarkadi, S.:

Lies, bullshit, and deception in agent-oriented programming languages. In: Proceedings of the 20th International Trust Workshop Co-located with AAMAS/IJCAI/ECAI/ICML (AAMAS/IJCAI/ECAI/ICML 2018) (2018)

[31] Walton, D.: Abductive Reasoning, p. 320. University of Alabama Press, Tuscaloosa, AL (2014)

[32] Josephson, J.R., Josephson, S.G.: Abductive Inference: Computation, Philosophy, Technology, p. 316. Cambridge University Press, Cambridge, UK (1994)

[33] Flach, P.A., Kakas, A.C. (eds.): Abduction and Induction: Essays on Their Relation and Integration. Springer, Berlin Heidelberg (2000). https://doi.org/10.1007/978-94-017-0606-3

[34] Kakas, A., Kowalski, R., Toni, F.: Abductive logic programming. Journal of Logic and Computation **2**(6), 719–770 (1993). https://doi.org/10.1093/logcom/2.6.719

[35] Denecker, M., Kakas, A.C.: Abduction in logic programming. In: Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part I, pp. 402–436. Springer, Berlin, Heidelberg (2002)

[36] Denecker, M., de Schreye, D.: Sldnfa: An abductive procedure for abductive logic programs. The Journal of Logic Programming **34**(2), 111–167 (1998). https://doi.org/10.1016/S0743-1066(97)00074-5

[37] Ray, O., Kakas, A.: Prologica: a practical system for abductive logic programming. In: Proceedings of the 11th International Workshop on Non-monotonic Reasoning, pp. 304–312 (2006)

[38] Fung, T.H., Kowalski, R.: The iff proof procedure for abductive logic programming. The Journal of Logic Programming **33**(2), 151–165 (1997). https://doi.org/10.1016/S0743-1066(97)00026-5

[39] Sadri, F., Toni, F.: Abduction with negation as failure for active and reactive rules. In: Lamma, E., Mello, P. (eds.) AI*IA 99: Advances in Artificial Intelligence, pp. 49–60. Springer, Berlin, Heidelberg (2000)

[40] Azzolini, D., Bellodi, E., Ferilli, S., Riguzzi, F., Zese, R.: Abduction with probabilistic logic programming under the distribution semantics. International Journal of Approximate Reasoning **142**, 41–63 (2022). https://doi.org/10.1016/j.ijar.2021.11.003

[41] Bard, N., Foerster, J.N., Chandar, S., Burch, N., Lanctot, M., Song, H.F.,

Parisotto, E., Dumoulin, V., Moitra, S., Hughes, E., Dunning, I., Mourad, S., Larochelle, H., Bellemare, M.G., Bowling, M.: The hanabi challenge: A new frontier for AI research. Artificial Intelligence **280**, 103216 (2020). https://doi.org/10.1016/j.artint.2019.103216

[42] Siu, H.C., Peña, J.D., Chang, K.C., Chen, E., Zhou, Y., Lopez, V.J., Palko, K., Allen, R.E.: Evaluation of human-ai teams for learned and rule-based agents in hanabi. CoRR **abs/2107.07630** (2021) https://arxiv.org/abs/2107.07630

[43] O'Dwyer, A.: Quuxplusone/Hanabi: Framework for writing bots that play Hanabi. https://github.com/Quuxplusone/Hanabi/ (2017)

[44] Osawa, H.: Solving hanabi: Estimating hands by opponent's actions in cooperative game with incomplete information. In: AAAI Workshop: Computer Poker and Imperfect Information (2015). http://aaai.org/ocs/index.php/WS/AAAIW15/paper/view/10167

[45] Cox, C., Silva, J.D., Deorsey, P., Kenter, F.H.J., Retter, T., Tobin, J.: How to make the perfect fireworks display: Two strategies for Hanabi. Mathematics Magazine **88**(5), 323–336 (2015). https://doi.org/10.4169/math.mag.88.5.323

[46] van den Bergh, M.J.H., Hommelberg, A., Kosters, W.A., Spieksma, F.M.: Aspects of the cooperative card game hanabi. In: Bosse, T., Bredeweg, B. (eds.) BNAIC 2016: Artificial Intelligence, pp. 93–105. Springer, Cham (2017)

[47] Walton-Rivers, J., Williams, P.R., Bartle, R., Perez-Liebana, D., Lucas, S.M.: Evaluating and modelling hanabi-playing agents. In: 2017 IEEE Congress on Evolutionary Computation (CEC), pp. 1382–1389 (2017). https://doi.org/10.1109/CEC.2017.7969465

[48] Hu, H., Lerer, A., Peysakhovich, A., Foerster, J.: "Other-play" for zero-shot coordination. In: III, H.D., Singh, A. (eds.) Proceedings of the 37th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 119, pp. 4399–4410. PMLR, Virtual event (2020). https://proceedings.mlr.press/v119/hu20a.html

[49] Lerer, A., Hu, H., Foerster, J., Brown, N.: Improving policies via search in cooperative partially observable games. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 7187–7194 (2020). https://doi.org/10.1609/aaai.v34i05.6208

[50] Foerster, J., Song, F., Hughes, E., Burch, N., Dunning, I., Whiteson, S., Botvinick, M., Bowling, M.: Bayesian action decoder for deep multi-agent reinforcement learning. In: Chaudhuri, K., Salakhutdinov,

R. (eds.) Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 97, pp. 1942–1951. PMLR, Long Beach, CA (2019). https://proceedings.mlr.press/v97/foerster19a.html

[51] Sarmasi, A., Zhang, T., Cheng, C.-H., Pham, H., Zhou, X., Nguyen, D., Shekdar, S., McCoy, J.: Hoad: The hanabi open agent dataset. In: Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems. AAMAS '21, pp. 1646–1648. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2021)

[52] Rao, A.S.: AgentSpeak(l): BDI agents speak out in a logical computable language. In: Lecture Notes in Computer Science, pp. 42–55. Springer, Berlin, Heidelberg (1996). https://doi.org/10.1007/bfb0031845

[53] Shapiro, S.S., Wilk, M.B.: An analysis of variance test for normality (complete samples). Biometrika **52**(3-4), 591–611 (1965). https://doi.org/10.1093/biomet/52.3-4.591

[54] Ross, A., Willson, V.L.: Paired Samples T-Test, pp. 17–19. SensePublishers, Rotterdam (2017). https://doi.org/10.1007/978-94-6351-086-8_4

[55] Kullback, S., Leibler, R.A.: On information and sufficiency. The Annals of Mathematical Statistics **22**(1), 79–86 (1951). https://doi.org/10.1214/aoms/1177729694

[56] Panisson, A.R., Ștefan Sarkadi, McBurney, P., Parsons, S., Bordini, R.H.: On the formal semantics of theory of mind in agent communication. In: Agreement Technologies, pp. 18–32. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17294-7_2

[57] Harbers, M., Bosch, K.v.d., Meyer, J.-J.: Modeling agents with a theory of mind. In: 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology, vol. 2, pp. 217–224 (2009). https://doi.org/10.1109/WI-IAT.2009.153