Modelling Human-Like Decision Making and Social Trust Using Probabilistic Programming

Maciej Olejnik

St Edmund Hall

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Hilary 2022

Acknowledgements

First and foremost, I would like to thank my supervisor, Prof. Marta Kwiatkowska for her patience and not giving up on me when I was close to giving up myself. Without her guidance and feedback, the completion of this thesis would not have been possible. Further, I would like to thank Wendy and my family for believing in me and supporting me throughout this bumpy journey of completing a doctorate. I am also grateful to everyone who helped shape my experiment for proving invaluable feedback. Finally, I would like to thank the EPSRC Doctoral Training Partnership for covering my course fees, and the University of Oxford along with DeepMind for awarding me the Oxford-DeepMind Graduate Scholarship, which provided financial support during my studies.

This work received partial support from the FUN2MODEL project (grant agreement No. 834115) led by my supervisor Marta Kwiatkowska, funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme.

Abstract

Effective collaborations between humans and machines necessitate the modelling of human cognitive processes and complex social attitudes such as trust, guilt or shame. Robots can take an active role in reducing misuse if they are able to detect human biases, inaccurate beliefs or overtrust, and make accurate predictions of human behaviour.

To that end, we propose cognitive stochastic multiplayer games, a novel parametric framework for multi-agent human-like decision making, which aims to capture human motivation through mental, as well as physical, goals. Our framework enables expression of cognitive notions such as trust in terms of beliefs, whose dynamics is affected by agent's observation of interactions and own preferences. Agents are modelled as soft expected utility maximisers, which allows us to capture the full range of rationality, from imperfections characteristic of human rationality, to fully rational robots. A key contribution is a novel formulation of the utility function, which incorporates agent's own, as well as other agents', emotions and takes into account their preference over different goals. Heuristics and mental shortcuts that people use to approximate what they cannot observe are captured in the framework as mental state estimation functions.

We implement the model using a probabilistic programming language called WebPPL. Our tool supports encoding of cognitive models and simulating their execution based on stochastic behavioural predictions it generates. Conversely, given a set of data, the tool may be used to learn characteristics of agents using Bayesian techniques. The software has been designed to be modular, so that probabilistic models of affection developed by others may be integrated. We validate our tool on a number of synthetic case studies, demonstrating that cognitive reasoning can explain experimentally-observed human behaviour that standard, equilibria-based analysis often overlooks.

To evaluate the framework in a human-robot interaction setting, we have designed and conducted an experiment that has human participants playing the Trust Game against a custom bot we developed. Participants in the game (humans or bots) are randomly assigned the role of an investor or investee. Results of our study show that predictions of human behaviour generated by our tool are on par with, and in some circumstances superior to, the state of the art. Unlike other approaches, our model integrates behavioural observations with prior beliefs and captures how one agent's behaviour affects actions of their opponent.

Contents

List of Figures

List of Abbreviations

AI	Artificial intelligence.
ASMAS	Autonomous stochastic multi-agent system.
BDI	Beliefs, desires, intentions.
BSCC	Bottom strongly connected component.
CSMG	Cognitive stochastic mulitplayer game.
CTL	Computational tree logic
DTMC	Discrete-time Markov chain
DRA	Deterministic Rabin automaton.
GASP	Guilt and shame proneness.
GSS	General Social Survey.
LTL	Linear temporal logic.
MCMC	Markov chain Monte Carlo.
MDP	Markov decision process.
ML	Machine learning.
MSE	Mean squared error.
MSPE	Mean squared probabilistic error.
NE	Nash equilibrium.
PCTL	Probabilistic computational tree logic.
PDF	Probability density function.
PMF	Probability mass function.
POMDP	Partially observable Markov decision process.
PPL	Probabilistic programming language.
RNN	Recurrent neural network.
SCC	Strongly connected component.
SEU	Subjective expected utility.
SMC	Sequential Monte Carlo.
SMG	Stochastic multiplayer game.

1
Introduction

Contents

The desire of humans to develop automated machines, or automata, can be traced back to Ancient Greece and China. Those predecessors of today's robots often resembled humans or animals and were powered by water, steam or air pressure. However, it was not until the second half of the twentieth century and the emergence of industrial robots that automation began to significantly enhance human productivity [1]. By the end of the century, widespread use of automated systems in aviation, information retrieval, fire control and navigation has led researchers to consider its negative aspects and ways in which humans misuse automation [2].

Presently, fuelled by rapid advancement in artificial intelligence (AI), we are entering a new chapter of human-robot interactions, characterised by close partnerships and cooperation. Progress in machine learning promises to make robots truly autonomous and put them on equal footing with humans. This transition has the potential to transform the society in many positive ways. Home-assistive robots may improve care of the elderly [3]; connected autonomous vehicles are expected to greatly reduce the number of traffic accidents [4]; utilising AI-based image recognition has been shown to improve diagnostic accuracy in skin cancer detection [5]; and autonomous unmanned aerial vehicles show promise for enhancing post-disaster search and assessment [6].

However, with great power comes great responsibility. Stuart Russell, an influential researcher in the field, argues AI poses existential threat to humanity [7]. While that claim is debatable [8], it is hard to ignore real-life examples of dangers posed by autonomous machines. Numerous fatal crashes involving Tesla vehicles in autopilot mode have been widely reported in the media [9–11]. Overtrust in robots has been put forward as one of the causes of these accidents [12]. Considering mental models of the drivers has been suggested as a way to prevent future failures of autonomous vehicles [13]. Sheng et al. [14] propose how trust can be taken into account when planning routes for automated vehicles. Another prominent topic is the use of AI in military applications. If used appropriately, autonomous technology can revolutionise warfare and save many lives. But first, troubling ethical issues must be resolved, especially in light of recent reports of unmanned combat aerial vehicles autonomously targeting humans for the first time [15, 16].

Ensuring smooth and safe collaboration between humans and robots will require machines to understand our motivations and emotions and be able to mimic them [17–21]. Affective computing [22] concerns itself with studying and developing computational systems that understand human affects. It has recently been suggested that probabilistic programming, which facilitates expression of stochastic models and making inferences by supporting probabilistic constructs as part of a programming language, is highly appropriate for expressing affective models [23]. It turns out that psychological theories of emotion can often be formulated as generative models and implemented as probabilistic programs, which are modular and easy to adopt, often compatible with deep learning libraries, and naturally express uncertainty present in these models.

The work presented in this thesis aims to be one of the building blocks of successful human-robot partnership of the future. We combine insights from affective computing with standard methods of game theory to put forward a novel model of human-like decision making intended for use in robots. Our formalism draws from the theory of mind [24] by carefully representing what humans do, and what they do not, know. Moreover, we utilise findings from cognitive sciences and behavioural decision theory to capture limitations in reasoning of *homo sapiens*. Our model is highly parametric, reflecting the wide, continuous spectrum of human personalities. Finally, the framework is computational in nature, and we provide a probabilistic programming implementation to validate the theory.

An emotion that we are particularly interested in is trust. Often described as the glue of a healthy society, trust is widely recognised for facilitating economic growth of societies [25], improving health and well-being of communities [26] and fostering better governance [27]. It is generally agreed that appropriately calibrating trust of human users towards robots is vital in reducing misuse [28]. Hence, a prominent feature of our decision-making model is a formal, novel definition of trust, inspired by theoretical work of psychologists and political scientists.

1.1 Thesis Outline

We now give a brief overview of how the work described in the rest of this document came about and what is to be expected in the subsequent chapters. The original goal of our research was to formalise social trust and provide mechanisms for reasoning about it. The first attempt at this task resulted in a framework called autonomous stochastic multi-agent system (ASMAS) [29], described in detail in Chapter 4, to which the author of this thesis contributed. ASMAS expresses trust as a logical operator and provides model checking algorithms to evaluate trust formulae. However, scrutiny of this formalism (see Section 4.6) reveals its fundamental weaknesses and inappropriateness for use in robots.

To overcome these deficiencies, we set out to develop a more intuitive, implementable, data-driven framework. With formalisation of trust in mind, we had taken a step back and considered what motivates humans and why they prefer one course of action over another. This resulted in a more general model, capable of expressing not only trust, but also other mental attitudes, such as guilt, satisfaction or surprise. We call it cognitive stochastic multiplayer game (CSMG) – a computational framework that models human-like decision-making process and generates behaviour predictions. Its theoretical side is described in Chapter 5, whereas Chapter 6 describes our implementation of the framework and a selection of case studies.

To validate this novel model, we have conducted a human study based on a famous money-exchange scenario called trust game, details of which can be found in Chapter 7. Finally, Chapter 8 identifies strengths and weaknesses of our work and suggests possible directions of future research.

1.2 Contributions

Our high-level objective throughout this work has been to facilitate future interactions of humans with autonomous robots. We postulate that machines must understand people's motivations and emotions for our relationships with robots to thrive. This insight led us to investigate human decision-making process and the way it is affected by emotions, with special emphasis on trust. In particular, the main contributions of this thesis are:

- Model checking algorithms for ASMAS (Section 4.5), which are based on PCTL* model checking techniques, the key difficulty being the history-dependence of ASMAS; it is overcome by considering a bounded subset of the logic and computing finite prefixes that refute or validate a given formula.

- Cognitive stochastic multiplayer game – an emotion-aware model of human-like decision making (Chapter 5). It integrates existing behavioural theories in a computational framework that aims to truthfully represent limitations of human reasoning.

- A novel definition of trust formulated in the setting of CSMG (Section 5.4.4) – our model is designed to allow expression of a variety of mental attitudes, but trust plays a special role in the context of human-robot partnerships.

- A probabilistic programming implementation of the CSMG model that supports learning agent preferences from behavioural data (Chapter 6) – our choice of paradigm follows a recent trend in affective computing, enabling easier integration of existing models of emotion into our tool.

- An experiment with human subjects that evaluates the predictive power of our framework and tests various hypotheses about human-robot partnerships (Chapter 7) – what sets our study apart is the sophisticated decision-making mechanism of the robot, time horizon of the interaction and the varying role of the machine.

2 Related Work

Contents

This chapter sets the scene for describing our contributions with an overview of relevant literature. We begin by summarising most notable models and logics of trust in Section 2.1. We then give a synopsis of the problem of predicting decisions of humans in Section 2.2. This fundamental issue has been studied across many disciplines; in this chapter we focus on computational approaches that are relevant in the context of our work. However, note that Section 3.5 supplements this with a more general and high-level overview of the matter. Finally, in Section 2.3 we outline human experiments based on the trust game [30] scenario carried out to date, with special focus on those that feature robots as game participants.

2.1 Reasoning about Trust

Trust is an overloaded term, studied in a variety of disciplines and taking different meanings depending on the context in which it is used. However, we delay discussing such philosophical issues to Section 3.1; instead, below we overview the most notable attempts to express and reason about trust in the context of computer science.

2.1.1 Trust Model Types

Given the importance of trust for developing and preserving relationships between agents, it should not come as a surprise that there exist several formal models of this concept. Most of them, however, are specific to a particular domain and incorporate a non-cognitive notion of trust. The type of trust they express generally falls into one of two categories: credentials-based, adopted in security, where a user gains access to a resource by supplying appropriate credentials; or experience-based, where agent's trustworthiness is determined from past interactions. Examples include: (i) a Bayesian network model applicable to peer-to-peer networks (and possibly other distributed systems) where trust is a proportion of successful interactions to total interactions [31], (ii) a swarm dynamics model where trust is assumed to simply mirror performance [32], and (iii) a formalisation which models trustworthiness of information sources using argumentation theory [33].

Of more interest from the point of view of this thesis are models developed to capture human's trust towards a robot, or, more generally, automation – we term this third type cognitive trust. An early example of such work models temporal evolution of trust of human operators of semi-automatic machines [34]. Authors use autoregressive moving average vector form of time series analysis to obtain a differential equation that describes how operator's trust depends on the performance and fault rate of automation. Recent approaches have focused on the setting of humans and robots cooperating as part of the same team and the arising need for appropriate trust. For example, Xu and Dudek [35] propose OPTIMo which captures (i) the causal reasoning of how robot's performance affects human's trust and (ii) evidential factor analysis that infers people's trust levels based on their actions with a Bayesian network. Floyd et al. [36] present an algorithm for trustworthiness-aware robot action selection that uses case-based reasoning methodology. Schaefer et al. [37] divide the trust process into three parts: (i) trust development, (ii) trust outcomes and (iii) trust calibration. The authors perform a meta-analysis to investigate trust formation and conclude that human-related factors, particularly to do with emotions, as well as capability of automation have an effect on trust-development.

Chen at al. [38] also consider how a model of human trust may be leveraged to improve task performance of a human-robot team, but their approach is different. Trust is modeled as a (discrete) latent variable whose evolution is described by a linear Gaussian system. Supplementing that with a model of human behaviour produces what authors call trust-POMDP (Partially Observable Markov Decision

Process), characterised by the fact that execution histories are approximated by trust values when synthesising strategies.

An aspect common to all the above models is that only a single value of trust between a human and a robot is assumed to exist, regardless of what task is being performed. Soh et al. [39] address that limitation by representing trust as a (latent) function, rather than a variable, and modelling how it transfers between different tasks. A neural model (RNN) and a Bayesian Gaussian process model are proposed to capture how trust transfers between tasks and experimental evidence is given that suggests both can be useful depending on the application.

However, this and other models mentioned above are limited to human-robot cooperation scenarios and only focus on one side of that relationship, where human is the trustor and robot the trustee. Crucially, lack of intentionality on the part of the machine is assumed, which reduces human's trust to an assessment of robots's competence. Wagner et al. [40] take a step back by proposing a conceptual framework of human-robot trust that uses game-theoretic representations. In particular, the authors consider games in normal form and put forward a mechanism for action selection of the trustor based on the relationship between their risk aversion and the risk associated to choosing the trusting action (computed based on the payoff matrix of the game). Substantial experimental evidence is presented that tests the hypotheses generated by the framework; some of the findings indicate that the model is oversimplistic. Indeed, the formalism offers little in the way of concreteness and authors themselves admit its main objective is to inspire future research.

2.1.2 Logics for Trust

Before we turn to logical languages designed to reason about trust, we first review some formalisms that capture human attitudes. In particular, we mention the so-called BDI logics, which are based on the theory of an American philosopher Michael Bratman. In his influential book [41], he explores human decision-making processes and identifies beliefs, desires and intentions (BDI) as its crucial components. He argues that, while desires influence our actions only potentially, intentions control our conduct directly and are essential for understanding the practical reasoning of humans. Bratman's theory gave rise to a number of modal logics [42–44]. They extend existing temporal or propositional logics with operators for reasoning about mental attitudes, whose semantics is given in terms of certain accessibility relations defined on the underlying transition system. Building on the above and using Castelfranchi & Falcone's theory [45], Herzig and Lorini propose a BDI-like logic for reasoning about trust and reputation [46]. They successfully capture the cognitive notion of trust by defining a trust operator in terms of agent's goals and beliefs. However, their logic is qualitative in nature and hence insufficient to accurately express quantitative trust.

2.2 Predicting Human Behaviour

Modeling decision making of humans is a universal problem, studied in a variety of disciplines – its thorough review could easily outgrow this thesis. We split our treatment of this topic into two parts. In Section 3.5, we outline how this problem has been approached by computer scientists through a historical lens. In this section, we take a more focused look at a selection of works that are relevant to the solution we propose.

As described in Section 3.5, the first breakthrough in the modelling of human behaviour involved the introduction of a utility function to model our preferences. This quickly led to expected utility theory, which became the standard way of resolving human behaviour. However, many researchers argued for inappropriateness of this model [47–49], which ultimately culminated in the very influential prospect theory, described briefly in Section 3.5, and its successor, cumulative prospect theory, which alters the weight function to operate on cumulative, rather than absolute, probabilities.

Certain behavioural models diverge from prospect theory; one such, called BEAST [50], postulates sensitivity to expected values and to the probability of experiencing regret (and four additional but less important mechanisms) dictates human decisions. Other frameworks incorporate behavioural insights into machine learning algorithms; an example is provided by a procedure dubbed psychological forest [51], which uses handcrafted features based on theorised decision-making mechanisms and predictions generated by BEAST as inputs to a learning procedure.

In any case, all those models are aimed at a particular setting common to choice prediction competitions, namely, a (often binary) choice between lotteries. Indeed, even though cumulative prospect theory was designed to handle decision nodes of high multiplicity, it remains specialised to well-defined choices between wealth-resembling quantities.

An alternative method of adapting traditional game theory to reflect human intricacies involves proposing a novel formulation of the utility function. The most prominent example are psychological games [52], which extend a standard concept of a game by assuming that utility of agents depends not only on outcomes, but also on beliefs that agents hold about the future behaviour of their opponents.

This allows one to model how emotions of players influence their behaviour, with a restriction that such emotions must be expressible in terms of one's initial (possibly nested) belief. However, psychological games, while admitting solution methods based on standard game-based techniques such as backward induction and equilibria computation, assume fixed payoff structure, do not support inference of beliefs from data or belief updating, and have no associated software implementation.

Dynamic psychological games [53] address some of the limitations of the original proposal by including a more robust belief structure and supporting non-equilibrium analysis. However, a hierarchy of deeply nested beliefs is problematic from the computational point of view. Moreover, the model ignores uncertainty and imperfect information that humans face when making choices, and it does not support learning.

2.3 Trust Game Experiments

Trust game is a simple, money-exchange scenario proposed by Berg et al. [30] (see Section 4.4 for details). It is generally accepted as a practical way to measure trust based on behaviour, as opposed to self-reporting which is considered less reliable. There have been over one hundred instances of trust game experiments with humans performed in the past; see Johnson et al. [54] for a meta-analysis. The general conclusion is that humans are trusting, trustworthy and cooperative.

Recently, some researchers set out to study trust of humans toward machines by instantiating trust game with people and robots. We have identified three publications where such experimental setup arises [55–57] (referred to below as Oksanen, Schniter and Mota, respectively). Of those, two (Oksanen and Schniter) are similar in that participants play a single-shot trust game with a machine in a non-physical form (i.e., "bots") and the main aim is to determine whether humans trust machines differently than they trust humans.

In particular, Oksanen introduces six different types of opponents that vary in how they are described ("robot", "artificial intelligence" or no description) and how they are called ("Michael" or "jdrx894"). The experiment is run with a large sample size ($N = 1077$), where each participant is randomly allocated to one of six conditions and asked to specify an investment amount (out of initial $1,000). However, participants were not incentivised monetarily to select an optimal amount as their remuneration was flat. The findings are that average investments are similar in all conditions, but they are highest when the opponent is described as a "robot named jdrx894". The main conclusion is that opponent type has no statistically significant effect on trust. Trust is also found to be positively correlated with technical education, online robot exposure and robot use self-efficacy.

Schniter reach similar conclusions, though their setup is slightly different. They consider three conditions: (i) a human playing against another human, (ii) a human playing against a robot and (iii) a human playing against a robot where any profit of the robot is transferred to another human. Importantly, unlike Oksanen, who do not consider the robot's strategy and do not record the amount returned by the machine, Schniter endow the robot with the following behaviour (which participants are made aware of): the return amount is randomly selected from the set of returns observed in condition (i) (human vs human). Participants play a single round of the trust game with their opponent and their remuneration reflects their earnings in the game. However, even though participants are motivated to carefully consider their investment, the behaviour of the robot is over-simplistic and by assumption human-like. In any case, the findings are consistent with Oksanen.

Finally, Mota differs from the above in that a physical, humanoid robot is used, sample size is very small ($N = 5$) and two iterations of trust game are considered. The robot is operated using a Wizard-of-Oz algorithm, i.e., controlled by a human. A simple trust model consisting of an initial assessment and an iterative update is used. It is found that trusting robots is similar to trusting other humans, in a sense that the model underlying trust remains the same. However, authors observe that it is more difficult for humans to form an initial assessment due to lack of social clues from the machine. It is proposed that people fill this gap by either leveraging or creating social experience.

In summary, all three experiments (i) involve robots that are not autonomous, (ii) put the human in the role of the investor, and (iii) feature a very limited horizon of the interaction. In contrast, in this thesis we describe a human study that features an autonomous bot whose behaviour is driven by a sophisticated decision-making mechanism, playing repeated trust game against humans who take either role in the game.

3 Background

Contents

In this chapter, we give an overview of the standard theory that this thesis builds upon. Any attempt to formalise trust must be preceded by a review of what psychologists, sociologists and political scientists have already established; this is summarised in Section 3.1. The models we propose are stochastic and probability theory features heavily in this thesis; an overview of the most relevant aspects can be found in Section 3.2. One of the models we propose is grounded in the theory of formal verification, outlined in Section 3.3. Analysing interactions of self-interested agents has traditionally been the domain of game theory; relevant concepts, which the other model we put forward builds upon, are introduced in Section 3.4. Even though humans are generally self-interested, purely game-theoretic approaches often fail to account for our behaviour. As a result, a significant body of literature exists on computational approaches to modelling human decisions; this is outlined in Section 3.5.

3.1 Trust Theory

We begin by giving an overview of trust itself; we quote the most noteworthy definitions of this concept and mention its conjectured properties; we point at ways in which trust can be measured, and list its attributes that have been verified experimentally.

3.1.1 Cognitive Theory of Trust

Trust is inherently a common-sense, intuitive notion, used in a variety of disciplines, and defined in multiple ways. One of the first attempts at formalising trust came in 1967 from an American psychologist Julian Rotter, who defined it as an "expectancy held by an individual that the word, promise or written communication of another can be relied upon" [58]. Another classic definition, by Diego Gambetta, formulates trust as a "subjective probability with which an agent assesses that another agent will perform a particular action, both before he can monitor such action and in a context in which it affects his own action" [59]. Yet another definition, in fact one of the most commonly accepted, states that trust is a "willingness of a party to be vulnerable to the actions of another party based on the expectation that the other party will perform a particular action important to the trustor, irrespective of the ability to monitor or control that other party" [60]. The above formalisations provide a taste of the diversity of trust definitions; depending on who you ask, trust may take a form of a mental attitude, expectation, belief, willingness, intention or even behaviour. It is often characterised by vulnerability and risk on the trustor's side, and a particular action of a trustee.

An attempt to unite all the different definitions and provide a general formulation of trust has been undertaken by psychologists Cristiano Castelfranchi and Rino Falcone [45]. They consider trust to be "a mental state, a complex attitude of an agent A towards another agent B about the behaviour/action relevant for the result (goal) g". This definition requires A to be a *cognitive agent*, i.e. an agent endowed with goals and beliefs. It also emphasises that trust of agent A towards B must be considered relative to a goal; otherwise, it is generally meaningless. Furthermore, two crucial ingredients of the mental state of trust are identified: (i) a competence belief: A believes that B is capable of taking the required action, and (ii) a disposition belief: A believes that B is willing to execute the task. The authors postulate also that trust is a quantitative notion and that its value should be understood as a "subjective certainty of pertinent beliefs".

3.1.2 Experimental Trust Studies

It is generally agreed that measuring trust is difficult. Traditionally, two different methods have been utilised for that purpose - surveys and experiments. Early examples of questionnaires put forward to estimate subjects' propensity to trust include Rotter's Interpersonal Trust Scale [58], Rosenberg's Faith in People Scale [61], a trust scale by Rempel et al [62] or fragments of the General Social Survey[1]. They typically involve a few dozens of trust-related statements, to which participants associate a number from some small range (most commonly on a 5 or 7-point scale), expressing to what extent they agree with it. More specific surveys exist, which are applicable to a given area of research, to mention Muir's questionnaire [63] for automated systems, Human Computer Trust Rating Scale [64] for air traffic control or a human-robot interaction trust scale measure [65]. In fact, most experimental studies design their own questionnaires to measure specific, goal-related trust (e.g. [66, 67]).

That brings us to the second standard method of measuring trust, which is by observing actions performed by subjects in experiments. A classic example is provided by studies in the area of economics, in particular the "investment game" [30] (sometimes referred to as "trust game"). It involves two subjects A and B, first of whom receives \$10, which he or she may either keep, or send any proportion of it to B. Any amount A chooses to invest will be tripled in transit. Finally, upon receiving the sum, B may share any fraction of it with A. Contradictory to the analysis based on Nash equilibria, which predicts A to keep all the money, vast majority of human subjects decide to invest all, or some part, of their initial \$10 with the other player [30]. It is generally agreed that A's decision of whether to share the money with B involves trust, and indeed that the more A trusts B, the more he or she will invest. Glaeser et al [68] conducted an experimental study based on the investment game, which aimed at verifying whether trust questionnaires

[1]GSS is a sociological survey created and regularly collected since 1972 by the National Opinion Research Center at the University of Chicago.

predict trusting behaviour. Questions were partially sourced from existing trust surveys, such as that of Rosenberg and Rotter, or GSS, and partially designed by the authors. The results of the study indicate that attitudinal questions generally do not predict trusting behaviour, unless they are specific (in which case correlation exists, but is weak). On the other hand, inquiries about participants' past behaviour were found to positively correlate with trusting behaviour in the experiment.

Recently, a new way of measuring trust has emerged from the field of neuroscience. Functional brain imaging has been employed to assess brain activation accompanying trust-related tasks. A common experimental setup in that community involves subjects evaluating faces for trustworthiness [69]. Increased activity in brain regions called amygdala and right insula was detected when participants viewed faces they deemed untrustworthy [70]. On the other hand, judging faces trustworthy was associated with enhanced signal change in right superior temporal sulcus (STS). Another study, which used similar methodology, found that propensity to trust can be associated with increased grey matter volume in certain brain regions [71]. Krueger et al. [72] monitored brain activity of humans playing the investment game and identified two distinct brain regions, paracingulate cortex (PcC) and septal area (SA), which underlie decisions to trust in that scenario. In yet another study, Dimoka [73] found that brain activations associated with trust and distrust better predict bids submitted in online auctions than survey-based measures. Even though the results so far are promising and indicate brain imaging could provide an objective, unbiased measure of trust, low availability and high cost of fMRI scans remains a major obstacle for mainstream adoption of the technique.

An active area of experimental research on trust, and relevant from our point of view, is that of human-robot interaction. For example, Desai et al. [74] studied trust dynamics in presence of failures and found that periods of low reliability of a robot earlier in the interaction reduce trust more than similar periods later on, and furthermore, the recovery of trust in the former case is slower. However, the authors discovered also that the decline of trust may be significantly slowed down if the robot acknowledges its lack of ability by providing confidence feedback. Finally, trust recovery after a period of low reliability was found to be a slower process than normal development of trust in the absence of failures. Robinette et al. discovered that humans tend to overtrust robots in emergency evacuation scenarios [75]. In another study, trust repair was investigated, the conclusion being that promises and apologies can be effective in repairing trust, but only if performed at the right time; in particular, this needs to happen the next time reliance occurs, rather than immediately after a breach of trust [67]. Sheng et al. [76] combine self-reports with physiological measurements to study the dynamics of human trust in the setting of autonomous driving; they find that the pertinence of alarms that the vehicle emits has a significant effect on trust.

3.2 Probability

Probability features heavily in this thesis. All the systems we consider are probabilistic; our implementation uses probabilistic programming; various probability distributions are used to model agents' beliefs. Therefore, a probability primer is in order.

3.2.1 Probability & Measure Theory

The notion of probability has several interpretations, but the one most humans are comfortable with revolves around quantifying chances of some event happening. We evaluate the likelihood of rain on a given day, of getting a six in a roll of dice or winning a lottery. This intuition is formalised with a notion of a *probability space*, which is a triplet $(\Omega, \mathcal{F}, \mu)$, where Ω is the sample space (e.g., representing the set of outcomes of an experiment), $\mathcal{F}$ is the set of events (each event being a subset of Ω) and $\mu : \mathcal{F} \to [0, 1]$ is a probability measure that assigns probabilities to events. This construction has roots in measure theory, which devises methods of "measuring" sets. In fact, to ensure that the above definition satisfies basic, desirable properties, we require that $(\Omega, \mathcal{F})$ is a *measurable space*. That, in turn, requires that $\mathcal{F}$ is a σ-algebra on Ω, meaning that $\mathcal{F}$ is a collection of subsets of Ω that includes Ω itself and is closed under complement and countable unions. The collection $\mathcal{F}$ is often referred to as *measurable sets*. Moreover, for μ to be a *probability measure*, it must be countably additive (i.e., $\mu(\cup_{i=0}^{\infty} A_i) = \sum_{i=0}^{\infty} \mu(A_i)$ for a countable collection of pairwise disjoint sets $\{A_i\}_{i=0}^{\infty} \subseteq \mathcal{F}$) and satisfy $\mu(\Omega) = 1$. For more detail, we refer the reader to a standard textbook on measure theory, such as the one by Billingsley [77].

In practice, one way to go about constructing a probability space is to start with a set of outcomes Ω. One then selects subsets of Ω that are easy to assign probability to and one shows that such assignment satisfies some standard properties. Any collection of subsets of Ω that contains Ω itself can be extended to a σ-algebra. Finally, one uses one of standard theorems of measure theory to uniquely extend the partially defined probability assignment to a probability measure. Such construction is common for transition systems and an example is given in Section 3.3.1.

3.2.2 Bayes' Theorem

One of the most influential theorems in probability theory is, in fact, also a contender for the most basic one. It formalises the intuition of updating beliefs about the world based on observations. For example, imagine you throw a pair of dice and you are not allowed to see the outcome. With no additional information, you would be right to assume a pair of threes is as likely as a pair of sixes. However, had someone told you that the sum of dice outcomes is six, you would revise your expectations and discard the pair of sixes (and most of other outcomes) as a possibility.

This type of reasoning is formalised through the notion of *conditional probability*. Fix a sample space Ω, a σ-algebra $\mathcal{F}$ and a probability measure Pr. Given two events $A, B \in \mathcal{F}$ such that $\Pr(B) \neq 0$, the conditional probability of A given B is denoted $\Pr(A|B)$ and defined as $\frac{\Pr(A\cap B)}{\Pr(B)}$. Intuitively, given that B happened, the universe of possible outcomes shrinks to those included in B.

Bayes' Theorem relies on a very simple observation. As long as $\Pr(A) \neq 0$, we can apply the definition of conditional probability the other way round: $\Pr(B|A) = \frac{\Pr(B\cap A)}{\Pr(A)}$. From there, basic algebraic manipulation yields the renowned theorem:

$$\Pr(A|B) = \frac{\Pr(B|A)\Pr(A)}{\Pr(B)}.$$

This incredibly simple result laid a foundation for a successful branch of statistics (Bayesian Statistics) and it has infiltrated other areas of science, such as machine learning or cognitive science. The application that is the most relevant from our point of view is Bayesian inference, which postulates using Bayes' Theorem to revise beliefs having observed evidence. For example, assume you do not know which of your two friends, Bob or Alan, is faster. They are both sporty and have similar body shapes, so you think it is equally likely that either of them is faster than the other. In the nomenclature of Bayesian inference, this initial hypothesis is referred to as a prior. Then, you get to observe the two men participate in the same race and Alan's time is twenty seconds faster than Bob's. You may ask yourself: assuming Alan was the faster of the two, what are the chances that he would beat Bob by twenty seconds? Pretty high, probably. This, in Bayesian jargon, is called the likelihood (of evidence given the hypothesis). Given prior and likelihood, one uses Bayes' Theorem to compute the posterior, i.e., the updated belief.

3.2.3 Probability Distributions

Certain probability spaces are so important that they are given a name. This usually happens either because probability measure has desirable properties or because it is a very good approximation of some natural phenomena (or both). These "famous" probability spaces are usually referred to as probability distributions and it has become customary to use this somewhat vague term in place of more precise notions such as probability measure or probability space. Throughout this thesis, when we say "probability distribution on a set S", we refer to an assignment of probabilities to elements of some σ-algebra (defined in Section 3.2.1) $\mathcal{F}$ on S, where $\mathcal{F}$ itself is clear from the context. For example, when S is a finite set, $\mathcal{F} = 2^S$ and when S is an interval on $\mathbb{R}$, $\mathcal{F}$ is the Borel σ-algebra, i.e., the smallest σ-algebra that contains all the open intervals inside S. In what follows, for a set S, $\mathcal{D}(S)$ denotes the set of probability distributions on S. Further, the set of those distributions under which elements that are assigned a positive probability form a countable subset of S is denoted $\mathcal{D}^f(S)$. Also, for a distribution $d \in \mathcal{D}(S)$, $\text{supp}(d) = \{s \in S \mid d(s) > 0\}$ denotes the support of d.

Below, we briefly overview families of probability distributions that we make extensive use of in this work. Reflecting the nature of the sample space Ω, each probability distribution is either discrete or continuous. Discrete probability distributions are characterised by a probability mass function (PMF) $p : \Omega \to [0, 1]$, which assigns some mass to each point in the sample space. We require that $\sum_{\omega\in\Omega} p(\omega) = 1$. Continuous probability distributions are described by a probability density function (PDF) $f : \Omega \to \mathbb{R}$, which assigns relative likelihoods to elements (points) of Ω. We require that $\int_{\omega\in\Omega} f(\omega) = 1$.

For each family, we give the sample space Ω and the PMF or PDF, as appropriate.

Dirac Distribution Arguably the simplest of all probability distributions, Dirac distribution (also called δ-distribution) can be defined on any discrete sample space Ω and it assigns all probability mass to a selected element $\omega \in \Omega$. In other words, the Dirac distribution on a countable set S centred on $t \in S$ is given by a PMF p such that

$$p(s) = \begin{cases} 1 & \text{if } s = t, \\ 0 & \text{otherwise.} \end{cases}$$

Categorical Distribution Whenever the sample space Ω is a finite set, any probability distribution defined on Ω is an instance of the categorical distribution. Letting $n = |\Omega|$ and assuming an order on elements of Ω, a categorical distribution on Ω is given by a vector $\vec{p} = (p_1, \dots, p_n)$ of probabilities, where $\sum_{i=1}^{n} p_i = 1$.

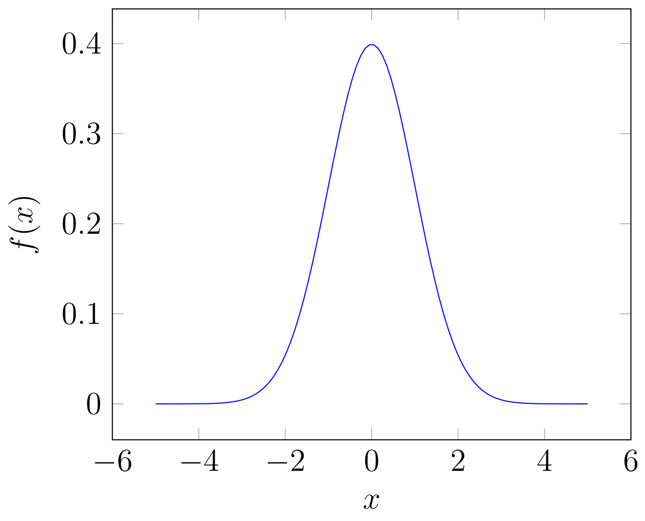


Figure 3.1: The PDF of a standard Gaussian distribution ($\mu = 0$, $\sigma = 1$)

Poisson Distribution Many real-life phenomena are characterised by events that happen randomly and independently, but with some constant long-term frequency. Examples include incoming calls in a call centre, people arriving at a bus stop or traffic accidents occurring on a given intersection. The probability of a given number of events happening in a fixed interval of time is described by a Poisson distribution, characterised by a parameter $\lambda > 0$, with a PMF $p : \mathbb{N} \to [0, 1]$ given by

$$p(k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

Gaussian Distribution The first family of continuous probability distributions we consider, and one that is very widely used in practice, is a Gaussian (or normal) distribution. It describes distributions of various statistics of populations, notably height, weight or intelligence. It is often described as a "bell curve" due to the shape of its PDF when plotted in Cartesian coordinates (see Figure 3.1).

The general form of the probability density function of a Gaussian distribution is

$$f(x) = \frac{1}{2\sqrt{\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2},$$

parameterised by its mean μ and standard deviation σ.

Uniform Distribution Another very common distribution is one that assigns the same probability mass/density (hence its name - uniform distribution) to all elements of sample space Ω. In the simplest case, when Ω is finite with $|\Omega| = n$, each element is assigned probability mass $\frac{1}{n}$, i.e., $p(\omega) = \frac{1}{n}$ for all $\omega \in \Omega$. Another case relevant for this thesis is when sample space Ω is an interval, $[a, b]$ say. Then the PDF of uniform distribution on Ω is a constant function $f(x) = \frac{1}{b-a}$ for all $x \in [a, b]$.

Dirichlet Distribution Finally, we describe the most complex distribution yet, which is not only continuous, but also multivariate, meaning that the sample space Ω is multi-dimensional. Specifically, $\Omega = \{(x_1, \ldots, x_K) \mid \sum_{i=1}^{K} x_i = 1\}$, where K is one of the parameters of a Dirichlet. The second parameter is a vector $\vec{\alpha} = (\alpha_1, \ldots, \alpha_K)$ with $\alpha_i > 0$ for $1 \leq i \leq K$. Given $\vec{\alpha}$, $\mathrm{Dir}(\vec{\alpha})$ is short for the Dirichlet distribution with parameter $\vec{\alpha}$ (with K given by the size of $\vec{\alpha}$). The intuition about $\vec{\alpha}$ is that α_i is the "weight" of the i^{th}dimension of Ω, in a sense that the greater α_i is, the greater will the value of x_i be in expectation. In fact, the expectation of the value of x_i under $\mathrm{Dir}(\vec{\alpha})$ is

$$\frac{\alpha_i}{\sum_{j=1}^{K} \alpha_j}.$$

The PDF of $\mathrm{Dir}(\vec{\alpha})$ is

$$f(x_1, \ldots, x_K) = \frac{1}{B(\alpha)} \prod_{i=1}^{K} x_i^{a_i - 1},$$

where $B(\alpha)$ is a normalising constant (needed to ensure PDF integrates to 1 over the sample space), whose definition is irrelevant for the purposes of this thesis.

3.2.4 Probabilistic Programming

Due to their usefulness for modelling uncertainty, probability distributions are commonly used in models of random or not fully understood processes, such as weather forecasting, predicting stock prices or, more recently, affective computing. A typical workflow in which a probabilistic model is utilised involves feeding observational data into it and generating predictions, be it a weather forecast, stock price or a human emotion.

Because of high complexity of such models, the only practical approach is to use computers to generate those predictions. One could employ a standard programming language for this purpose, but that would require implementing probability distributions and inference methods from scratch. Instead, many would opt for a probabilistic programming language (PPL) that supports probabilistic constructs out of the box.

Probabilistic programming languages typically build upon well-established languages and can be thought of as external libraries; notable examples include Pyro [78], built on top of Python, WebPPL [79], which extends a subset of Javascript, and Anglican [80], which is integrated with Clojure. Some PPLs are self-contained, most notably Stan. Any respectable probabilistic programming language provides two functionalities: (i) specification of probabilistic models and (ii) methods for making inferences, typically based on observed data, using a specified model.

Stochastic models typically encode our understanding of some random processes. For instance, flipping a fair coin may lead to two outcomes, heads or tails, both of which are equally likely. This can be encoded as a trivial probabilistic model; for example, Listing 1 shows how it may be implemented in WebPPL. It takes the form of a function which returns `'H'` or `'T'` with equal probabilities, achieved by using a predefined function `flip`, which samples from a uniform distribution over $\{\texttt{true}, \texttt{false}\}$.

```
let coin = function() {
  return flip() ? 'H' : 'T'
}
```

Listing 1: WebPPL model of a coin flip

A somewhat more involved, but still very basic, model is presented in Listing 2. It represents the process of drawing a random card from a shuffled deck. It uses a function `uniformDraw` that draws a sample from a uniform distribution over the elements of an array passed to it.

```
let drawCard = function() {
  let suit = uniformDraw([♥, ♠, ♣, ♦])
  let rank = uniformDraw(
   ['2','3','4','5','6','7','8','9','10','J','Q','K','A']
  )
  return rank + suit
}
```

Listing 2: WebPPL model of drawing a card from a shuffled deck

A more complex example of a process that affects human life on a daily basis is weather formation. Suppose you want to predict if it will rain on a given summer day. A reasonable approach would be to study historical data to obtain the base rate of rain depending on the month. However, suppose you have also observed that, when it is cloudy in the morning, it is more likely to rain later in the day. You could then formulate a probabilistic model such as the one presented in Listing 3, where the function `flip` is now passed an argument $p \in [0, 1]$ that specifies a probability that `flip` will return `true`.

Formulating stochastic models as probabilistic programs is useful, but in many cases one does not understand the underlying mechanism well. However, based on

```
let rain = function(month, clouds) {
    return
      (month == 'june' && flip(0.11)) ||
      (month == 'july' && flip(0.08)) ||
      (month == 'august' && flip(0.05)) ||
      (clouds && flip(0.3))
}
```

Listing 3: WebPPL model predicting if it will rain on a given day

observations and basic comprehension, certain causal links may be hypothesised to exist. Then, the power of probabilistic programming allows one to *infer* the model, provided enough data is available. As an example, consider the problem of identifying what leads to lung cancer. It is currently well understood that smoking is the primary cause of that disease, but genetics and exposure to certain chemicals, particularly radon, plays a role too. However, assuming these causal links were not known, probabilistic inference could help discover them.

Suppose several factors are hypothesised to contribute to lung cancer risk: smoking, history of cancer in one's family, exposure to radon and consumption of meat. To investigate the effect of each of these factors, we could record this data for a random (large enough) sample of deceased persons and feed it to a program such as the one from Listing 4. It uses WebPPL's `Infer` operator to compute the risk of lung cancer in two hypothetical scenarios. `Infer` performs marginal inference – the way it operates depends on the selected inference method (in this case, "enumerate"), but it always accepts a stochastic computation as input and returns a probability distribution on the return values of that computation.

Inference by enumeration involves exploring all possible execution paths of the provided stochastic model, each associated to a unique resolution of randomness in the model. Naturally, it is applicable only when there is a finite number of outcomes at each random choice. This is the case in our lung cancer risk modelling, since the data set is finite. To achieve the desired effect of computing cancer risk associated to a set of values of some or all factors, we use conditioning. It is captured by a special `condition` operator available in WebPPL, which can only be used in the scope of `Infer`, and has an effect of discarding all executions for which the specified condition (passed as argument to `condition`) does not hold. Therefore, the computation of lung cancer risk for a meat eater (line 14) is equivalent to iterating over the elements of `data` array, discarding those where `meatEater` is set to `false`, and returning the distribution of values of `cancer` among the remaining data points. Computing

```
 1  // stores all the recorded data
 2  let data = [
 3    {
 4      smokes: true,
 5      familyHistory: true,
 6      radonExposure: false,
 7      meatEater: true,
 8      cancer: true
 9    }
10    // ,...
11  ]
12
13  // Does eating meat affect lung cancer risk?
14  let meatEaterRisk = Infer({method: 'enumerate'}, function() {
15    let d = uniformDraw(data)
16    condition(d.meatEater)
17    return d.cancer
18  })
19
20  // Suppose Bob does not smoke and does not eat meat, but has
21  // a history of cancer in the family and high levels of radon have
22  // been detected in their house. What's Bob's lifetime risk of
23  // developing lung cancer?
24  let bobRisk = Infer({method: 'enumerate'}, function() {
25    let d = uniformDraw(data)
26    condition(d.smokes)
27    condition(d.familyHistory)
28    condition(d.radonExposure)
29    condition(d.meatEater)
30    return d.cancer
31  })
```

Listing 4: WebPPL model that infers lung cancer risk from data

Bob's risk of cancer (line 24) proceeds along the same lines, except that filtering of data points is more discriminating, reflecting the presence of more conditions.

Finally, we mention that, even though inference by enumeration computes an exact probability distribution, marginal inference is generally intractable and approximate methods, such as Markov chain Monte Carlo (MCMC) or sequential Monte Carlo (SMC), must often be employed. This happens either when there are so many possible executions of the stochastic model that enumerating them all is infeasible, or when sampling from continuous probability distributions is involved and enumeration is impossible.

3.2.5 Probabilistic Predictions

Section 3.2.4 showed how generative models can be encoded as probabilistic programs and used to make predictions. To evaluate the constructed model, we would like to compare its predictions with reality. For instance, the rain forecast generated by our model from Listing 3 should be confronted with our observation of whether it rained or not.

The measure we will use to evaluate such stochastic predictions, which are represented by discrete probability distributions, is called *mean squared prediction error* (MSPE), and it is a probabilistic version of a standard mean squared error (MSE). Suppose a probabilistic prediction over a finite set of outcomes Ω is represented by the probability mass function $p : \Omega \to [0, 1]$. Then, upon observing an outcome $\bar{\omega} \in \Omega$, the MSPE of p is given as

$$\mathrm{MSPE}(p) = \sum_{\omega \in \Omega} p(\omega)(\omega - \bar{\omega})^2.$$

Hence, MSPE(p) is an expected value of a mean squared error of a deterministic prediction selected according to p.

3.3 Automated Verification

Automated verification concerns itself with checking algorithmically whether given models satisfy specified properties. It has been applied to verify a variety of software, hardware or biological systems against specifications formulated in a logical language. This section provides an overview of the most common models, logics and verification techniques employed in this field. Note that we focus our attention on probabilistic verification, and so the models we consider are all stochastic.

3.3.1 Models

Markov Chain The models that lend themselves to (probabilistic) verification techniques are generally variations of (probabilistic) transition systems. One of the most basic such models, called *(discrete-time) Markov chain* (DTMC), is defined as a tuple $(\mathrm{S}, s_{\mathrm{init}}, \mathbf{T}, \mathrm{L})$, consisting of a finite set of states S, of which $s_{\mathrm{init}} \in \mathrm{S}$ is designated as the initial state, the transition probability matrix $\mathbf{T} : \mathrm{S} \times \mathrm{S} \to [0, 1]$ and a labelling function $\mathrm{L} : \mathrm{S} \to 2^{AP}$. An intuitive interpretation of a Markov chain is that of a machine that starts in the initial state s_{init} and repeatedly transitions to the next state according to $\mathbf{T}$. In particular, from state $s \in \mathrm{S}$, the probability of moving to another state $s' \in \mathrm{S}$ is $\mathbf{T}(s, s')$. Transitions are assumed to occur at discrete time steps and each state s is labelled with a subset $\mathrm{L}(s)$ of atomic propositions AP that hold in that state.

An execution of a Markov chain $\mathcal{M}$ is represented by an infinite path $\delta = s_0 s_1 s_2 \ldots$ such that $\mathbf{T}(s_i, s_{i+1}) > 0$ for all $i \geq 0$ and $s_0 = s_{\text{init}}$. A set of infinite paths of $\mathcal{M}$ is denoted $\mathrm{IPath}^{\mathcal{M}}$. Any non-empty prefix of an infinite path δ is a finite path and $\mathrm{FPath}^{\mathcal{M}}$ denotes the set of all finite paths. Moreover, for a finite path $\rho = s_0 s_1 \ldots s_n$ and an infinite path $\delta = t_0 t_1 \ldots$: (i) $|\rho|$ $(= n)$ denotes the length (i.e., number of transitions) of ρ, (ii) $\mathrm{last}(\rho)$ retrieves the last state of ρ, (iii) $\delta(i)$ $(= t_i)$ denotes the i^{th} state of δ, (iv) $\delta[0..m] = t_0 t_1 \ldots t_m$ denotes a finite prefix of δ of length m and (v) $\delta[m..\infty]$ denotes an infinite suffix of δ. Often, it is useful to consider paths that start in a given state $s \in \mathrm{S}$ which is not the initial state. The set of such paths is denoted $\mathrm{FPath}^{\mathcal{M}}(s)$ (finite) or $\mathrm{IPath}^{\mathcal{M}}(s)$ (infinite).

To verify properties of Markov chains, one generally has to reason about probabilities of reaching certain (appropriately labelled) states. This is made rigorous by defining a probability measure $\mathrm{Pr}_{\mathcal{M}}$ on the set of infinite paths $\mathrm{IPath}^{\mathcal{M}}$ of a DTMC $\mathcal{M}$. For a full construction of a probability space, we refer the reader to Kemeny et al. [81]; what follows is an intuitive description. The first step is to assign a probability to a *finite* path $\rho = s_0 s_1 \ldots s_n$ by multiplying transition probabilities along the path: $\mathbf{T}(\rho) = \mathbf{T}(s_0, s_1) \cdot \mathbf{T}(s_1, s_2) \cdot \ldots \cdot \mathbf{T}(s_{n-1}, s_n)$. With that, given a finite path ρ, a cylinder set $C(\rho) \in \mathrm{IPath}^{\mathcal{M}}$ is defined as a set of all infinite paths that have a prefix ρ: $C(\rho) = \{\delta \in \mathrm{IPath}^{\mathcal{M}} \mid \rho \text{ is a prefix of } \delta\}$. Based on that, a family of measurable sets $\Sigma_{\mathrm{IPath}^{\mathcal{M}}}$ is defined as the σ-algebra generated by the cylinder sets and the probability measure $\mathrm{Pr}^{\mathcal{M}}$ on $\Sigma_{\mathrm{IPath}^{\mathcal{M}}}$ is given as the unique measure that satisfies $\mathrm{Pr}^{\mathcal{M}}(C(\rho)) = \mathbf{T}(\rho)$ (existence and uniqueness is guaranteed by a standard theorem from measure theory, see e.g. Theorem 11.3 from [77]).

Finally, we mention that each Markov chain $\mathcal{M}$ has an underlying directed graph, whose nodes are states of $\mathcal{M}$, with an edge from s to s' present if $\mathbf{T}(s, s') > 0$. Standard graph-theoretical concepts apply to that graph; in particular, a strongly connected component (abbreviated SCC) of $\mathcal{M}$ denotes a set of states U such that for each pair of states (s, s') in U, there is a path from s to s' with all states in U, and U has no proper superset which satisfies that property. Further, a bottom SCC (BSCC in short) of $\mathcal{M}$ is an SCC U from which no state outside of U is reachable.

Markov Decision Process Markov chains are suitable for modelling a variety of natural phenomena and have found many applications in biology, chemistry or physics. However, they are inherently probabilistic and hence not powerful enough to express agency. For that, a *Markov decision process* (MDP) is introduced; it extends Markov chains with nondeterminism by equipping the model with a set of actions Act. A resulting tuple that defines MDPs is $(\mathrm{S}, s_{\mathrm{init}}, \mathrm{Act}, \mathrm{T}, \mathrm{L})$, where S, s_{init} and L are as before, but $\mathrm{T} : \mathrm{S} \times \mathrm{Act} \rightarrow \mathcal{D}(S)$ is now a (partial) probabilistic transition function. With MDPs, a transition can be thought of as a two-stage process: from state $s \in \mathrm{S}$, say, an action a is selected (nondeterministically), followed by a probabilistic change to the next state according to probability distribution $\mathrm{T}(s, a)$. Note that, given arbitrary $s \in \mathrm{S}$ and $a \in \mathrm{Act}$, $\mathrm{T}(s, a)$ may be undefined; with a slight abuse of notation, we define $\mathrm{Act}(s) = \{a \in \mathrm{Act} \mid \mathrm{T}(s, a) \text{ is defined}\}$ to be the set of actions available in s. Intuitively, different actions are available in different states. Note that a Markov chain is a special case of an MDP with a set of actions being a singleton.

Similarly as for Markov chains, formal reasoning about properties of MDPs requires a probability space to be defined. Again, we refer the reader to Forejt et al. [82] for formal treatment and present an intuitive view below. A notion of a path has to be adapted to include not only states, but also actions taken in each state. We refer to the set of infinite (resp. finite) paths in an MDP $\mathcal{M}$ as $\mathrm{IPath}^{\mathcal{M}}$ (resp. $\mathrm{FPath}^{\mathcal{M}}$) as before. To assign probabilities to paths, nondeterminism arising from actions must be resolved; this is achieved by an *action strategy* $\sigma : \mathrm{FPath}^{\mathcal{M}} \rightarrow \mathcal{D}(\mathrm{Act})$ that specifies a plan of action at each point of execution. Under a strategy σ, an MDP $\mathcal{M}$ reduces to a DTMC $\mathcal{M}^{\sigma}$, and its probability measure $\mathrm{Pr}^{\mathcal{M}^{\sigma}}$ can be used to assign probabilities to measurable sets of paths of $\mathcal{M}$ (under σ). The set of possible strategies is denoted Σ.

3.3.2 Logics

To allow for automatic verification of properties of models, one needs a formal language that can express them. It is customary to use the language of logic for this purpose. Each logic comes with a set of operators, a way to apply and combine them, and the meaning of each operator. Sentences of a logical language are made up of logical operators and atomic propositions and are referred to as *logical formulae.* There are two types of formulae: a state formula, usually denoted ϕ, that can be evaluated in a state (of a Markov chain or an MDP), and a path formula, denoted ψ, that applies to paths, rather than states.

CTL The first logic we consider is called Computational Tree Logic (CTL). Its syntax is given by:

$$\begin{array}{rcl} \phi & ::= & \text{true} \mid p \mid \neg\phi \mid \phi \wedge \phi \mid \forall\psi \mid \exists\psi \\ \psi & ::= & \bigcirc\phi \mid \phi\mathcal{U}\phi \end{array}$$

A CTL formula is a state formula ϕ constructed according to the rules of the above grammar. Besides standard propositional operators like negation $\neg$ and $\wedge$, CTL defines the universal quantifier $\forall$ and an existential quantifier $\exists$. Given a path formula ψ and a state s, quantifiers allow formation of a state formula by stipulating that ψ holds for all paths (in the case of $\forall$), or at least one (in the case of $\exists$) path, starting in s. There are two types of path formulae. One is constructed using the *next* operator $\bigcirc$, which expresses that a given state formula ϕ holds in the second (i.e., the next one after the initial one) state of the path it is being evaluated on. The second type has the *until* operator $\mathcal{U}$ as its principal connective, which, when applied to two state formulae ϕ_1 and ϕ_2 on a path δ, expresses that ϕ_1 holds in all the states of δ up to (and excluding) the first state in which ϕ_2 holds.

The intuitive explanation above is formalised by specifying the semantics of the logic, which gives meaning to every operator. Given a DTMC $\mathcal{M} = (\mathrm{S}, s_{\text{init}}, \mathbf{T}, \mathrm{L})$, a state s (resp. an infinite path δ) and a state formula ϕ (resp. a path formula ψ), we write $\mathcal{M}, s \models \phi$ (resp. $\mathcal{M}, \delta \models \psi$) to denote that ϕ holds in state s (resp. ψ holds on a path δ). The satisfaction relation is defined inductively as follows:

$$\begin{aligned}
&\mathcal{M}, s \models \text{true for all } s \in \mathrm{S},\\
&\mathcal{M}, s \models p \text{ if } p \in \mathrm{L}(s),\\
&\mathcal{M}, s \models \neg\phi \text{ if not } \mathcal{M}, s \models \phi,\\
&\mathcal{M}, s \models \phi_1 \wedge \phi_2 \text{ if } \mathcal{M}, s \models \phi_1 \text{ and } \mathcal{M}, s \models \phi_2,\\
&\mathcal{M}, s \models \forall\psi \text{ if } \mathcal{M}, \delta \models \psi \text{ for all } \delta \in \mathrm{IPath}^{\mathcal{M}}(s),\\
&\mathcal{M}, s \models \exists\psi \text{ if } \mathcal{M}, \delta \models \psi \text{ for some } \delta \in \mathrm{IPath}^{\mathcal{M}}(s),\\
&\mathcal{M}, \delta \models \bigcirc\phi \text{ if } \mathcal{M}, \delta(1) \models \phi,\\
&\mathcal{M}, \delta \models \phi_1\mathcal{U}\phi_2 \text{ if there exists } i \geq 0 \text{ such that } \mathcal{M}, \delta(i) \models \phi_2\\
&\qquad \text{and } \mathcal{M}, \delta(j) \models \phi_1 \text{ for all } 0 \leq j < i.
\end{aligned}$$

Note how state formulae are evaluated relative to a state $s \in \mathrm{S}$, while path formulae are evaluated on a path $\delta \in \mathrm{IPath}^{\mathcal{M}}$. Note also that when expressing satisfaction of a given formula ϕ in the initial state s_{init} of $\mathcal{M}$, we may omit the state in the satisfaction relation and write $\mathcal{M} \models \phi$.

LTL CTL is referred to as a *branching-time* logic, reflecting the tree-like structure of possible futures captured by the universal and existential quantifiers. An alternative approach, termed *linear-time*, stipulates a "single-track" development of events, where each point in time has a unique successor. This perspective is employed by Linear Temporal Logic, defined by the following grammar:

$$\psi ::= \text{true} \mid p \mid \neg\psi \mid \psi \wedge \psi \mid \bigcirc\psi \mid \psi\mathcal{U}\psi$$

Hence, all formulae in LTL are path formulae and the semantics of the operators is virtually the same as for CTL, but adapted to paths (see a standard textbook [83]). While the introduction of LTL may at first appear to serve no purpose, since it introduces no new operators, a closer look reveals that LTL is no less expressive than CTL[2]. The crucial difference is that arbitrary path formulae are allowed as arguments to $\bigcirc$ and $\mathcal{U}$ operators, permitting formulae such as $p\mathcal{U}\bigcirc q$ or $\bigcirc\bigcirc p$, which are not part of CTL's syntax.

PCTL* Finally, we look at a logic that combines CTL with LTL and adds probability, called PCTL*. Its syntax is described by the following grammar:

$$\phi ::= \text{true} \mid p \mid \neg\phi \mid \phi \wedge \phi \mid \mathtt{P}^{\bowtie q}\psi,$$
$$\psi ::= \phi \mid \neg\psi \mid \psi \wedge \psi \mid \bigcirc\psi \mid \psi\mathcal{U}\psi.$$

The most important novelty of PCTL* is that quantifiers are replaced by the probability operator $\mathtt{P}^{\bowtie q}$ (where $\bowtie \in \{<, \leq, >, \geq\}$), which quantifies the probability of the set of paths starting in some state s satisfying some path formula. The second aspect that requires attention is greater freedom in forming formulae compared to CTL, afforded by allowing state formulae to be used where a path formula would normally be expected.

Specifying the semantics of PCTL* is, by and large, a matter of combining semantics of CTL and LTL, with the exception of the probability operator, which requires special treatment. To express the probability of satisfying a path formula ψ starting in a state s of a DTMC $\mathcal{M}$, we leverage the probability measure $\Pr^{\mathcal{M}}$ to define

$$Prob_{\mathcal{M},s}(\psi) = \Pr^{\mathcal{M}}(\{\delta \in \text{IPath}^{\mathcal{M}}(s) \mid \mathcal{M}, \delta \models \psi\}).$$

[2]It is also no more expressive than CTL. The expressiveness of the two logics is incomparable.

With that, given a DTMC $\mathcal{M} = (\mathrm{S}, s_{\mathrm{init}}, \mathbf{T}, \mathrm{L})$, a state $s \in \mathrm{S}$, state formulae ϕ, ϕ_1 and ϕ_2 and a path formula ψ, the satisfaction relation $\models$ for state formulae is as follows:

$$\begin{aligned}
&\mathcal{M}, s \models \text{true for all } s \in \mathrm{S},\\
&\mathcal{M}, s \models p \text{ if } p \in \mathrm{L}(s),\\
&\mathcal{M}, s \models \neg\phi \text{ if not } \mathcal{M}, s \models \phi,\\
&\mathcal{M}, s \models \phi_1 \wedge \phi_2 \text{ if } \mathcal{M}, s \models \phi_1 \text{ and } \mathcal{M}, s \models \phi_2,\\
&\mathcal{M}, s \models \mathtt{P}^{\bowtie q}\psi \text{ if } \mathit{Prob}_{\mathcal{M},s}(\psi) \bowtie q,
\end{aligned}$$

whereas for path formulae ψ_1 and ψ_2, and a path $\delta \in \mathrm{IPath}^{\mathcal{M}}$, the satisfaction relation is given by:

$$\begin{aligned}
&\mathcal{M}, \delta \models \phi \text{ if } \mathcal{M}, \delta(0) \models \phi,\\
&\mathcal{M}, \delta \models \neg\psi \text{ if not } \mathcal{M}, \delta \models \psi,\\
&\mathcal{M}, \delta \models \psi_1 \wedge \psi_2 \text{ if } \mathcal{M}, \delta \models \psi_1 \text{ and } \mathcal{M}, \delta \models \psi_2,\\
&\mathcal{M}, \delta \models \bigcirc\psi \text{ if } \mathcal{M}, \delta[1..\infty] \models \psi,\\
&\mathcal{M}, \delta \models \psi_1 \mathcal{U} \psi_2 \text{ if there exists } i \geq 0 \text{ such that } \mathcal{M}, \delta[i..\infty] \models \phi_2\\
&\qquad \text{and } \mathcal{M}, \delta[j..\infty] \models \psi_1 \text{ for all } 0 \leq j < i.
\end{aligned}$$

Finally, we note that the above generalises easily to MDPs. However, care needs to be taken to adapt the computation of the probability of a formula, since nondeterminism must be resolved to apply the probability measure. This is typically resolved by assuming implicit universal quantification over strategies. For example, formula $\mathtt{P}^{>q}\psi$ holds if and only if the probability of satisfying ψ on a future path is greater than q for *any* action strategy σ. Therefore, for an MDP $\mathcal{M}$ with a set of strategies Σ, we define

$$\begin{aligned}
\mathit{Prob}^{\mathit{min}}_{\mathcal{M},s}(\psi) &= \inf_{\sigma\in\Sigma} \mathrm{Pr}^{\mathcal{M}^\sigma}(\{\delta \in \mathrm{IPath}^{\mathcal{M}^\sigma}(s) \mid \mathcal{M}^\sigma, \delta \models \psi\}),\\
\mathit{Prob}^{\mathit{max}}_{\mathcal{M},s}(\psi) &= \sup_{\sigma\in\Sigma} \mathrm{Pr}^{\mathcal{M}^\sigma}(\{\delta \in \mathrm{IPath}^{\mathcal{M}^\sigma}(s) \mid \mathcal{M}^\sigma, \delta \models \psi\}),
\end{aligned}$$

so that the satisfaction relation for the probability operator is given by:

$$\mathcal{M}, s \models \mathtt{P}^{\bowtie q}\psi \text{ if } \mathit{Prob}^{\mathit{opt}(\bowtie)}_{\mathcal{M},s}(\psi) \bowtie q,$$

where

$$\mathit{opt}(\bowtie) = \begin{cases} \min & \text{when } \bowtie \in \{\geq, >\},\\ \max & \text{when } \bowtie \in \{\leq, <\}.\end{cases}$$

3.3.3 Model Checking

Now for the interesting part. Given a DTMC $\mathcal{M} = (\mathrm{S}, s_{\text{init}}, \mathbf{T}, \mathrm{L})$ and a PCTL* formula ϕ, how does one check whether $\mathcal{M} \models \phi$?[3] The process of answering that question is known as model checking and very efficient methods of doing it have been developed. We describe them below, starting with the very intuitive CTL model checking algorithm, followed by more obscure LTL model checking and culminating in PCTL* model checking techniques. Note that our treatment of the subject throughout this section is by necessity informal – rigorous approach would easily consume half of this thesis. We refer the reader to an excellent textbook by Baier and Katoen [83] for proofs, derivations and formal definitions.

CTL The basic idea for checking whether a CTL formula ϕ is satisfied in an initial state s_{init} of a DTMC $\mathcal{M}$ involves recursively computing satisfaction sets of subformulae of ϕ until the satisfaction set of ϕ itself can be computed. In particular, every CTL formula, say ϕ, is obtained by a repeated application of the grammar given in Section 3.3.2. Reversing this process gives the parse tree of ϕ, whose nodes are subformulae of ϕ (the root being ϕ itself) and leaves are atoms, such as true or p. To model check ϕ, we traverse its parse tree bottom-up, at each node computing all states in which a subformula ϕ_1 that this node corresponds to is satisfied (called the satisfaction set of ϕ_1). For the leaves of the parse tree, this task is trivial. For the nodes, the way the satisfaction set is computed depends on the principal connective of the subformula that the node represents; we overview the possible cases below. Once the satisfaction set H of subformula ϕ_1 is computed, the corresponding node may be turned into a leaf by introducing a new atomic proposition, which is true precisely in states that belong to H. That way, every node can be treated uniformly.

We now consider the crucial ingredient of the algorithm, i.e., the recursive step that computes the satisfaction set of a formula ϕ given satisfaction sets of its subformulae. We overview the key points of the algorithm, but for a detailed, rigorous treatment, we refer the reader to a standard textbook on model checking [83].

The course of action depends on the principal connective of ϕ. When this top-level operator is a standard propositional one, such as $\wedge$ or $\neg$, computing the satisfaction set of ϕ boils down to performing a set complement or intersection. The interesting case is when the principal connective is a quantifier, $\forall$ or $\exists$. Below we focus on the latter case, i.e., CTL formulae of the form $\exists \bigcirc \phi_1$ or $\exists(\phi_1 \mathcal{U} \phi_2)$.

[3]Recall that $\mathcal{M} \models \phi$ is short for $\mathcal{M}, s_{\text{init}} \models \phi$

$\phi = \exists \bigcirc \phi_1$ This is the simpler case of the two. Intuitively, states satisfying ϕ are states that have a successor state which satisfies ϕ_1. Formally, given a DTMC $\mathcal{M} = (\mathrm{S}, s_{\text{init}}, \mathbf{T}, \mathrm{L})$, $\mathrm{Sat}(\exists \bigcirc \phi_1) = \{s \in \mathrm{S} \mid \mathrm{Post}(s) \cap \mathrm{Sat}(\phi_1) \neq \emptyset\}$, where $\mathrm{Post}(s) = \{s' \in \mathrm{S} \mid \mathbf{T}(s, s') > 0\}$ is the set of successors of s in $\mathcal{M}$. This identity yields a simple algorithm for computing the satisfaction set of ϕ: iterate over states of $\mathcal{M}$ and, for each, check if it has a successor that belongs to $\mathrm{Sat}(\phi_1)$.

$\phi = \exists(\phi_1 \mathcal{U} \phi_2)$ Due to the "infinite" semantics of the $\mathcal{U}$ operator, this case is somewhat more involved. Computing the satisfaction set of ϕ takes the form of an iterative procedure inspired by an equivalence $\exists(\phi_1 \mathcal{U} \phi_2) \equiv \phi_2 \vee (\phi_1 \wedge \exists \bigcirc \exists(\phi_1 \mathcal{U} \phi_2)$. Intuitively, a state satisfies ϕ if either (i) it satisfies ϕ_2 or (ii) it satisfies ϕ_1 and has a successor state that satisfies ϕ. This gives rise to an iterative algorithm that starts with a set $U = \{s \in \mathrm{S} \mid s \in \mathrm{Sat}(\phi_2)\}$ and at each step adds a state $s' \in \mathrm{S}$, such that (i) $s' \notin U$, but (ii) ϕ_1 is satisfied at s' and s' has a successor in U.

The algorithms presented above can be easily adapted to handle CTL formulae whose principal connective is a universal quantifier.

LTL Unfortunately, LTL formulae do not admit a simple, intuitive model checking algorithm, as CTL formulae do. Instead, verifying an LTL formula ψ is reduced to checking a reachability (computing the probability of paths that reach a state in some target set) or persistence (computing the probability of paths that reach and forever remain in some target set) property, or computing the probability of accepting runs in a certain product Markov chain.

A key insight that makes this procedure work is that every LTL formula can be represented by a deterministic Rabin automaton (DRA). In particular, given a set of atomic propositions AP and an arbitrary LTL formula ψ, one can associate to it a DRA $\mathcal{A}$ with alphabet $\Sigma = 2^{AP}$, such that words accepted by $\mathcal{A}$ are precisely the infinite paths satisfying ψ. Now, to model check ψ on a Markov chain $\mathcal{M}$, one constructs a DRA $\mathcal{A}$ corresponding to $\neg\psi$, followed by forming a product Markov chain $\mathcal{M} \otimes \mathcal{A}$. Intuitively, $\mathcal{M} \otimes \mathcal{A}$ simulates execution of $\mathcal{M}$, additionally recording the corresponding state of $\mathcal{A}$. With that, $\mathcal{M} \models \psi$ is equivalent to nonexistence of a path in $\mathcal{M} \otimes \mathcal{A}$ satisfying the accepting condition of $\mathcal{A}$ (Algorithm 1).

Besides verifying an LTL formula ψ, it is often of interest to compute the probability that ψ is satisfied on paths starting from a given state, i.e., to verify formulae of the form $\mathtt{P}^{\bowtie q}\psi$. This requires computation of accepting BSCCs in the product Markov chain, along with their reachability probabilities. Then, the probability of ψ being satisfied in $\mathcal{M}$ is the same as the probability of reaching an accepting BSCC in $\mathcal{M} \otimes \mathcal{A}$ (Algorithm 2).

Algorithm 1: LTL model checking algorithm [83]

```
  input  : Markov chain $\mathcal{M}$, LTL formula $\psi$
  output : $\mathcal{M} \models \psi$
1 $\mathcal{A} \leftarrow$ DRA corresponding to $\neg\psi$;
2 Construct a product Markov chain $\mathcal{M} \otimes \mathcal{A}$;
3 if $\mathcal{M} \otimes \mathcal{A}$ has a path satisfying accepting condition of $\mathcal{A}$ then
4     return NO
5 else
6     return YES
7 end
```

Algorithm 2: Computing probability of an LTL formula [83]

```
  input  : Markov chain $\mathcal{M}$, state $s$, LTL formula $\psi$
  output : $Prob(s, \psi)$
1 $\mathcal{A} \leftarrow$ DRA corresponding to $\psi$;
2 Construct a product Markov chain $\mathcal{M} \otimes \mathcal{A}$;
3 Compute (accepting) BSCCs of $\mathcal{M} \otimes \mathcal{A}$;
4 $F \leftarrow \cup\{s \mid s$ is in some accepting BSCC$\}$;
5 $q_s \leftarrow \delta(q_0, L(s))$;                // state of $\mathcal{A}$ corresponding to $s$
6 return $Prob^{\mathcal{M}\otimes\mathcal{A}}((s, q_s), \Diamond F)$;
```

PCTL* We now give an overview of PCTL* model checking, which builds upon the techniques presented above. Formally, we consider the problem of, given a Markov chain $\mathcal{M}$ and a PCTL* formula ϕ, deciding whether $\mathcal{M} \models \phi$ holds. The main procedure is similar to the one for CTL; it involves a bottom-up traversal of the syntax tree of ϕ, which computes at each node the satisfaction set $\text{Sat}(\phi') = \{s \in S \mid s \models \phi'\}$, where ϕ' is the state subformula of ϕ corresponding to that node. Once $\text{Sat}(\phi')$ is determined, ϕ' is replaced in the original formula by a new atomic proposition a'_ϕ, such that $a'_\phi \in L(s)$ iff $s \in \text{Sat}(\phi')$.

As for CTL, the way $\text{Sat}(\phi')$ is computed depends on its principal connective and is trivial when it is a standard propositional operator. The interesting case is when $\phi' = \texttt{P}^{\bowtie q}\psi$; it turns out we can use the technique presented above for computing probabilities of LTL formulae.

Now, ψ will not in general be an LTL formula; however, with the replacement specified above, it will. Intuitively, what causes a path formula ψ in PCTL* to not be an LTL formula is the presence of the probability operator(s) P. Suppose $\phi' = \texttt{P}^{\bowtie q}\psi'$ is the innermost subformula of ψ that uses the P operator. Since ψ' has no P operator in its scope (by assumption), it is an LTL formula, whose probability can be computed according to Algorithm 2. Upon replacing ϕ' with an atomic proposition $a_{\phi'}$, we proceed recursively, eventually obtaining ψ with no P operators inside it.

This idea is formalised with a notion of *maximal proper state subformula* of a PCTL* formula ϕ, defined as a subformula of ϕ that differs from ϕ and is not contained in any other proper state subformula of ϕ. Then, if we encounter a formula of the form $\mathtt{P}^{\bowtie q}\psi$ as part of the model checking algorithm, we know that all maximal proper state subformulae in ψ have been replaced by atomic propositions; this follows from the bottom-up nature of the algorithm. Therefore, ψ is an LTL formula and the satisfaction set of $\mathtt{P}^{\bowtie q}\psi$ can be computed using automata-based approach. Algorithm 3 formalises the procedure outlined above.

Algorithm 3: PCTL* model checking algorithm [83]

```
input  : Markov chain $\mathcal{M}$, PCTL* formula $\phi$
output : Sat($\phi$)
1  $subs \leftarrow$ subformulae of $\phi$ ordered by formula length non-decreasingly;
2  foreach $\phi' \in subs$ do
3      switch $\phi'$ do
4          case $true$: $\mathrm{Sat}(\phi') = S$;
5          case $p$: $\mathrm{Sat}(\phi') = \{s \in S \mid p \in L(s)\}$;
6          case $\neg\phi''$: $\mathrm{Sat}(\phi') = S \setminus \mathrm{Sat}(\phi'')$;
7          case $\phi'_1 \vee \phi'_2$: $\mathrm{Sat}(\phi') = \mathrm{Sat}(\phi'_1) \cup \mathrm{Sat}(\phi'_2)$;
8          case $\forall\psi$: $\mathrm{Sat}(\phi') = \mathrm{Sat}_{LTL}(\psi)$ where $\mathrm{Sat}_{LTL}(\psi)$ is determined by
             LTL model checker;
9          case $\mathrm{Pr}_{\bowtie q}\psi$: $\mathrm{Sat}(\phi') = \{s \in S \mid Pr(s \models \psi) \bowtie q\}$ where $Pr(s \models \psi)$
             is determined by LTL model checker;
10     end
11     replace $\phi'$ by $a_{\phi'}$ in $\phi$ where $a_{\phi'} \in L(s)$ iff $s \in \mathrm{Sat}(\phi')$;
12 end
13 return Sat($\phi$);
```

3.4 Game Theory

Game theory concerns itself with mathematical formalisation of interactions between entities (be it people, computer agents, states or corporations) that pursue their own interests (which may or may not be aligned). The work of Oskar Morgenstern, John von Neumann [84] and John Nash [85], among others, has brought rapid progress of this area in the latter half of the 20thcentury. It is now widely used in economical models and has found a wealth of applications in computer science. Below, we overview the basic building blocks of game theory.

		Player 2	
		D	C
Player 1	D	$(-2, -2)$	$(0, -3)$
	C	$(-3, 0)$	$(-1, -1)$

(a) Prisoner's dilemma

		Player 2	
		H	T
Player 1	H	$(1, -1)$	$(-1, 1)$
	T	$(-1, 1)$	$(1, -1)$

(b) Matching pennies

Figure 3.2: Basic one-shot games

3.4.1 Normal Form Games

The most basic game structure, called a *normal form game*, is a tuple (N, Σ, u), where $N = \{1, \ldots, n\}$ is a set of players, $\Sigma = \{\Sigma_1, \ldots, \Sigma_n\}$ is a set of possible (pure) strategies of each player and $u = \{u_1, \ldots, u_n\}$ is a set of utility functions of players, where $u_i : \Sigma_1 \times \ldots \times \Sigma_n \to \mathbb{R}$ for $i \in N$. The interpretation of such a structure is a game in which each agent selects a single strategy, but their utility depends also on what other players choose.

A tuple of strategies, one for each player, $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n) \in \Sigma_1 \times \ldots \times \Sigma_n$ is called a *strategy profile*. Each strategy profile defines a unique outcome of the game. We also denote by $(\vec{\sigma_{-i}}, \sigma'_i)$ a strategy profile obtained by replacing the i$^{\text{th}}$component of $\vec{\sigma}$ with σ'_i.

Since each agent selects their action once, normal form games are often referred to as single-shot games. Additionally, when $n = 2$, i.e., there are only two players, a single-shot game can be represented by its *payoff matrix*, examples of which are displayed in Figure 3.2. Strategies are represented as names of rows and columns, such as D and T, while utilities of each player corresponding to each outcome make up the entries of the matrix. Utilities are often referred to as payoffs of the players.

Nash Equilibrium

The crucial question to ask about games such as those from Figure 3.2 is what actions should players take to maximise their utility.

Pure-Strategy NE This matter is addressed by one of the most celebrated concepts of game theory, that of Nash Equilibrium (NE). In its most basic form, it distinguishes those outcomes of the game that are characterised by the property stating that no player has an incentive to deviate from the chosen strategy. For example, in the prisoner's dilemma (Figure 3.2a), outcomes (D, C) and (C, D) both satisfy that property. Formally, a *(pure-strategy) Nash Equilibrium* is a strategy profile $\vec{\sigma}$ such that, for every player $i \in N$ and a strategy $\sigma'_i \in \Sigma_i$, $u_i(\vec{\sigma}_{-i}, \sigma'_i) \leq u_i(\vec{\sigma})$.

Mixed Strategy NE A natural question to ask at this stage is whether every game features a Nash Equilibrium. Thanks to a famous result by Nash [86], we can answer in the affirmative, as long as we generalise the above definition to *mixed strategies*. To see why this is necessary, consider the game of matching pennies, depicted in Figure 3.2b. Given any (pure) strategy profile, one of the players would rather switch their strategy and obtain a higher payoff. Therefore, no pure-strategy Nash Equilibrium exists. However, allowing mixed strategies, defined as probability distributions over pure strategies, enables us to recover an equilibrium for such games. In particular, equipping each player with a mixed strategy $\sigma = \langle H \rightarrow 0.5, T \rightarrow 0.5 \rangle$ that plays H and T with equal probabilities results in a Nash Equilibrium.

3.4.2 Extensive Form Games

Many scenarios that call for strategic reasoning are not as simple as deciding on a strategy and observing the outcome. More often, players repeatedly choose their actions, make observations and revise plans. Most board games, diplomacy or military conflicts follow this mechanism. As long as such a repeated game is finite, it can be represented as a normal form game. It is much more practical, however, to use an alternative representation, called *extensive form game*. While normal form is synonymous with a (payoff) matrix, extensive form amounts to a tree.

Syntax A tree structure $T = (V, E, v_0)$ consists of a set of vertices V, a set of edges $E \subseteq V \times V$ and a root $v_0 \in V$. The leaves (or terminal nodes) of T, denoted leaves$(T) \subseteq V$ are nodes with no outgoing edges, i.e., leaves$(T) = \{v \in V \mid (v, v') \notin E \text{ for all } v' \in V\}$.

In its simplest form, an extensive form game $\mathcal{G}$ amounts to a tuple $(N, T, \{V_i\}_{i \in N}, u_i)$, where $N = \{1, \dots, n\}$ is a set of players, $T = (V, E, v_0)$ is a tree structure, $\{V_i\}_{i \in N}$ is a partition on $V \setminus$ leaves(T) that assigns each node to one player and u_i : leaves$(T) \rightarrow \mathbb{R}$ represents payoffs of player $i \in N$. Nodes of the tree are states of the game, while edges are actions that players take. We assume that a single player takes action in every state (i.e., the game is turn-based), hence the partitioning of V into sets V_i. For convenience, each edge (v, v') is labelled with an action $a(v, v')$ and, for $v \in V_i$, $A(v) = \{a(v, v') \mid (v, v') \in E\}$ denotes the set of actions available to player i in v. Moreover, $A_i = \bigcup_{v \in V_i} A(v)$ consists of all actions available to a given player in all states that belong to them. An example of an extensive form game is depicted in Figure 3.3; note how edges are labelled with actions, and the partition of non-terminal game states is represented by colouring the nodes (black for player 1, white for player 2).

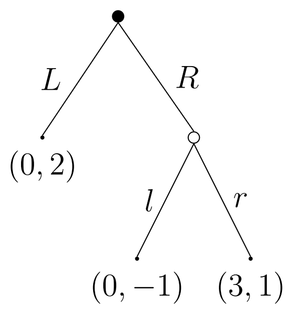


Figure 3.3: A two-player extensive form game

Note that, if we pick some non-terminal node $v \in V$ of a game $\mathcal{G}$, the subtree rooted at v represents a game in itself. Its components are easily defined by restriction to the subtree and such a game is referred to as a subgame of $\mathcal{G}$. For example, game from Figure 3.3 has a single subgame rooted at the white node.

The basic model just presented comes in many varieties, depending on the setting to which it is applied. Some major considerations are: (i) whether all players know the mechanics of the game (games of complete vs incomplete information), (ii) whether players can observe the current state of the game (imperfect vs perfect information games), (iii) if there is randomness in the way the game develops, or (iv) whether players can remember what happened in the past (perfect recall). For the time being, we assume the basic variation of extensive form game as stated, but we discuss most notable variations in Section 3.4.3.

Strategies A (pure) strategy $\sigma_i : V_i \to A_i$ of a player i dictates the action this player takes on their turn at any point of the game. As before, Σ_i denotes the set of pure strategies of i and Σ the set of strategy profiles. A mixed strategy is a probability distribution over pure strategies; however, as long as perfect recall is assumed, every mixed strategy $\alpha \in \mathcal{D}(\Sigma_i)$ has an equivalent behavioural strategy $\beta : V_i \to \mathcal{D}(A_i)$. This is significant because behavioural strategies are much more intuitive and easier to specify. Therefore, in what follows, we restrict our attention to behavioural strategies.

Equilibria Naturally, the concept of Nash Equilibrium transfers over to extensive form games. In fact, assuming perfect information, and thanks to the game being turn-based, a pure-strategy Nash Equilibrium is guaranteed to exist for every extensive form game. It can be computed using a very simple procedure called Zermelo's algorithm, or simpler, backward induction. The idea is to start at the leaves and make one's way up the tree, at each node selecting an optimal (i.e., one that maximises utility) action for the player whose turn it is. Such a node may then be replaced by the leaf corresponding to the chosen action. For example, applying backward induction to the game from Figure 3.3 yields a Nash Equilibrium $\langle R, r\rangle$ (where the strategy of each player is succinctly represented as the action they take at their unique decision node).

However, as we all know, "there ain't no such thing as a free lunch". In this case, guarantee of existence of a (pure-strategy) Nash Equilibrium comes with a cost of often having too many Nash Equilibria, especially when most of them display undesirable properties. Even in our simple example, closer inspection reveals that the strategy profile $\langle L, l\rangle$ is also a Nash Equilibrium. However, it relies on a non-credible threat of player 1 selecting action l, which is not in their best interest. Examples such as this one led Reinhard Selten to refine the notion of equilibrium with the introduction of *subgame-perfect equilibria* [87]. It achieves the goal of filtering unwanted equilibria by requiring that a strategy profile remains a Nash equilibrium when restricted to any subgame of the game in question.

Further problems arise when one starts considering variations of extensive form games, such as imperfect (or incomplete) information games. This lead various authors to formulate further equilibria refinements, notably sequential equilibria [88] or trembling hand perfect equilibria [89].

3.4.3 A Few Game Types

As mentioned above, many variations of extensive form, as well as normal form, games arise once one starts relaxing some of the assumptions. This section overviews the variations that are relevant from the point of view of this thesis.

Stochastic Games

Stochastic multiplayer games (SMG) generalise the concept of extensive form games by dropping the requirement of the state graph being a tree, and allowing probabilistic transitions and concurrency. Stochastic games share many features with transition systems introduced in Section 3.3.1 and can be considered a model bridging the gap between automated verification and game theory. A crucial novelty, compared to DTMCs and MDPs, is presence of multiple agents, whose preferences will be captured with a utility function. Due to unlimited time horizon and absence of terminal nodes, payoffs in stochastic games are associated to taking actions and visiting states. In particular, we define a stochastic multiplayer game (with rewards) as a tuple $(\mathrm{Ags}, \mathrm{S}, \{\mathrm{Act}\}_{A\in\mathrm{Ags}}, \mathrm{T}, \mathrm{R})$, where Ags is a (finite) set of agents; S is a (finite) set of states; $\mathrm{Act_A}$ is a (finite) set of actions of agent A; $\mathrm{T} : \mathrm{S} \times \mathrm{Act} \to \mathcal{D}(S)$ is a (partial) probabilistic transition function, where $\mathrm{Act} = \bigtimes_{A \in \mathrm{Ags}} \mathrm{Act_A}$ is a set of global actions; and $\mathrm{R} = \bigcup_{i=1}^{k} \{\mathrm{R_i}\}$ (for some $k \in \mathbb{N}$) is a set of reward structures, with $\mathrm{R_i} = \{\mathrm{r_{i,A}}\}_{A \in \mathrm{Ags}}$ and $\mathrm{r_{i,A}} = (\mathrm{r^{\mathbf{a}}_{i,A}}, \mathrm{r^{\mathbf{s}}_{i,A}})$, $\mathrm{r^{\mathbf{a}}_{i,A}} : \mathrm{S} \times \mathrm{Act} \to \mathbb{R}$ (action rewards), $\mathrm{r^{\mathbf{s}}_{i,A}} : \mathrm{S} \to \mathbb{R}$ (state rewards).

Turns Recall that each global action a is a vector of local actions of each agent; we use a_A to denote the component of a corresponding to A's local action. If an agent does not perform an action at a given state, we set $a_A = \perp$. A game $\mathbb{G}$ is *turn-based* if, for all $s \in \mathrm{S}$, there exists $A \in \mathrm{Ags}$ such that if $\mathrm{T}(s, a)(s') > 0$ for some a and s', then $a_B = \perp$ for all agents $B \neq A$. If that condition does not hold, then the game is called *concurrent*. We introduce a function $\textsc{actions} : \mathrm{S} \to \mathcal{P}(\mathrm{Act})$, which retrieves actions available in a given state, defined by $\textsc{actions}(s) = \{a \in \mathrm{Act} \mid \mathrm{T}(s, a) \text{ is defined}\}$. Also, for turn-based games, we introduce a function $\textsc{owns} : \mathrm{S} \to \mathrm{Ags}$, which associates a state s to the unique agent that takes an action in s, so $\textsc{owns}(s) = A$ iff there exists an $a \in \mathrm{Act}$ such that $\mathrm{T}(s, a)$ is defined and $a_A \neq \perp$.

Paths Let FPath denote the set of finite paths, which we take to be sequences of states interleaved with actions taken at those states, i.e., sequences of the form $s_0 a_0 s_1 \ldots s_n$, such that $\mathrm{T}(s_i, a_i)(s_{i+1}) > 0$ for all $0 \leq i < n$. We assume all paths start and end in a state. For a path $\rho \in \mathrm{FPath}$, we use $\mathrm{last}(\rho)$ to refer to the last state of ρ, $\mathrm{first}(\rho)$ to refer to its first state and $\mathrm{length}(\rho)$ is its length, defined as one less than the number of states in ρ. Moreover, if $\rho = s_0 a_0 s_1 \ldots a_{n-1} s_n$, then given $0 \leq i < j \leq n$, $\rho[i \ldots j]$ denotes a fragment $s_i a_i \ldots s_j$ of ρ. If $\delta \in \mathrm{FPath}$ is another finite path that satisfies $\delta[0 \ldots n] = \rho$, we say δ *extends* ρ or that ρ is a prefix of δ. We also use ρ^s to denote the set of states in ρ and ρ^a to refer to the set of actions taken along ρ. Finally, for $a \in \rho^a$, $s(a) \in \rho^s$ is the state at which a was taken (i.e., $s(a_i) = s_i$). We allow single-state paths of length 0.

Utility Given an agent $A \in \mathrm{Ags}$, cumulative utility function $\mathrm{u_A} : \mathrm{FPath} \to \mathbb{R}$ of A ranks possible paths according to agent A's preference, so that given ρ and ρ', $\mathrm{u_A}(\rho) > \mathrm{u_A}(\rho')$ if and only if A prefers ρ to ρ'. To reason about agent behaviour, it is helpful to assume a certain structure on the utility function; for example, taking it to be a linear combination of rewards is referred to as a linear utility. Then, given a set of coefficients $\{\lambda_i^A\}_{1 \leq i \leq k}$, utility gained by A in a state s is given by

$$\mathrm{u^{\mathbf{s}}_A}(s) = \sum_{i=1}^{k} \lambda_i^A \mathrm{r^{\mathbf{s}}_{i,A}}(s),$$

while utility obtained when taking action a at s is expressed as

$$\mathrm{u}^{\mathbf{a}}_{\mathrm{A}}(s,a)=\sum_{i=1}^{k}\lambda^{A}_{i}\mathrm{r}^{\mathbf{a}}_{\mathrm{i,A}}(s,a).$$

With that, cumulative utility gained by A along a path $\rho\in\mathrm{FPath}$ is given by

$$\mathrm{u}_{\mathrm{A}}(\rho)=\sum_{s\in\rho^{s}}\mathrm{u}^{\mathbf{s}}_{\mathrm{A}}(s)+\sum_{a\in\rho^{a}}\mathrm{u}^{\mathbf{a}}_{\mathrm{A}}(s(a),a).$$

Softmax Choice A benefit of numerical utility is that it allows one to define decision making in the face of uncertainty. It is common to assume that agents are soft expected utility maximisers. To clarify what that means, suppose that the system is in a state s and it is agent's A turn to take an action, selected from a set $A_0=\textsc{actions}(s)$. For $a\in A_0$, let U_a denote expected utility of A when they take an action a. Then, the probability of agent A taking some action $a_0\in A_0$ is

$$\frac{\exp(\alpha_A U_{a_0})}{\sum_{a\in A_0}\exp(\alpha_A U_a)},\tag{3.1}$$

where $\exp(x)=\mathrm{e}^{\mathrm{x}}$ denotes the Euler's number raised to power x. Thus, agents aim to maximise their expected utility, i.e., select an action that yields such a maximum, but do that with a certain amount of noise, measured by their rationality parameter $\alpha_A\in[0,\infty)$.

Partially Observable Games

A widely studied generalisation of games involves relaxing the assumption that all agents can observe the state of the game at all times. This is particularly justified in competitive scenarios where global state is made up of local states of each agent and an agent can't observe local states of its opponents. This imperfect information of players is typically captured by introducing for each agent a partition on the states of the game, referred to as information sets. The idea is that an agent cannot distinguish states belonging to the same information set. With that, care has to be taken that strategies as well as payoffs of players are defined in a way that does not allow an agent to distinguish between states in the same information set.

For games with large state spaces, explicitly specifying all information sets for each agent might not be practical. A more succinct representation is commonly used, whereby for each agent $A\in\mathrm{Ags}$, a set of observations O_A along with an observation function $obs_A:\mathrm{S}\to O_A$ is defined.

Bayesian Games

A second type of games characterised by players' lack of full information is referred to as games of incomplete information. Unlike imperfect information, which only concerns observability of game state, information incompleteness has to do with the ”rules“ of the game. In particular, in games of incomplete information, a player may not know the utility function of their opponent(s), the transition function, the available strategies or even who the other players are. Hence, this category of games is much more general than games of imperfect information and hence, much more difficult to solve. Through some clever observations, all sources of information incompleteness may be reduced to players not knowing each others' utility functions, which is what we assume below.

A major complication arising from this lack of information of players are infinitely nested expectations. Consider a two-person game in which players do not know each other's utility functions. To compute its course of action, player 1 will need to consider what their opponent may do, which in turn necessitates player 1 to consider player 2's utility, captured by player 1's first-order expectation. But then, player 2 may be tempted to estimate what player 1 thinks about player 2's motivation – this is already second-order expectation. This kind of reasoning may be continued *ad infinitum* and is highly problematic from the point of view of efficient representation and computation of solutions concepts.

An alternative model was famously proposed by John Harsanyi in his three-part paper [90–92]. His key contribution is a reduction of an incomplete information game to an imperfect information game, which, albeit different from the original, is game-theoretically equivalent to it. This is made possible by encoding the variability of possible characteristics of players with a set of types. Then, each configuration of player's utility function or any other variability in their attributes corresponds to a unique type of that player. Hence, assuming an assignment of types to players is given, the uncertainty in the game is eliminated. With that, the aforementioned reduction is achieved by assuming that initial transition of the game is preceded by a move of nature that assigns a type to each player, according to a probability distribution which is known to all players. Crucially, each player is assumed to know their own type, but cannot generally observe types of their opponents, which yields a game of imperfect information.

3.5 Modelling Human Decision Making

Human decision making has been studied by economists, psychologists, social scientists and philosophers for centuries [93]. An introduction of utility that captures human preferences has enabled expressing the ideas mathematically and led to expected utility maximisation principle, famously formalised by Von Neumann and Morgenstern [84], and its refinements, such as Subjective Expected Utility (SEU) [94]. However, numerous experiments involving human subjects have contradicted predictions made by this theory. As a result, many alternative formalisms have been devised to explain how humans make decisions.

Following a recent survey [95], modern models of human decision making can be divided into expert-driven, data-driven and hybrid. The first category comprises formalisms that are broadly based on expected utility principle, but modified to reflect limitations of human cognitive processes. For example, quantal response [96] assumes that agents compute expected utilities noisily and cognitive hierarchy [97] models dictate that humans can only perform nested reasoning up to a certain depth. Knowledge of a domain expert is usually required to devise an appropriate utility function and agents' parameters. Moreover, apparent deviations from rationality have been widely studied in behavioural sciences and a number of cognitive biases have been proposed based on results of experiments; this includes anchoring (over-reliance on single piece of information), herding (following majority), loss aversion, default bias (tendency to avoid decision making if default choice is provided) or framing (making different decision in the same situation depending on description).

Prospect theory [98] formalises some of the above observations in a mathematical model similar to expected utility theory. A crucial deviation from the standard method is that alternative outcomes are weighted according to an appropriately devised weight function, rather than their probabilities. This allows one to model experimentally observed deviations from rationality such as overweighting small probabilities or overweighting certain outcomes, resulting in a weight function that is subadditive in the lower region, subcertain ($\pi(p) + \pi(1-p) < 1$) and subproportional. The second major contribution of the authors is the value function which describes utility humans gain from changes in wealth or welfare. Informed by experimental evidence, it is proposed that the value function be concave for gains and convex for losses and steeper for losses than for gains. This reflects a commonly observed property of human preferences, which dictates that a loss of \$100 is felt more strongly than earning \$100, and the bigger the gain (or loss), the less sensitive we become to variations.

On the other hand, data-driven human decision-making models employ recent advances in machine learning (ML) to predict people's actions. Perhaps the most notable application was AlphaGo [99], a deep (and reinforcement) learning-fuelled AI agent that beat the best human players in the game of Go. Importantly, predicting moves of its opponent is a vital step in how AlphaGo computes its actions. Other notable data-driven models predict behaviour based on visual data, be it YouTube videos [100] or recordings of humans driving [101]. However, success of such models is dependent on training them with large amounts of data, which is not available for most applications. While technological progress that brings wearable devices into our lives is expected to alleviate that problem, widespread adoption is a matter of years rather than months.

As a result, high hopes are placed on the third, hybrid approach, which promises to combine the strengths and eliminate weaknesses of expert- and data-driven paradigms. It typically comprises a machine learning model in which features encode domain expert's knowledge and insight. Often, those additional features capture standard principles like utility maximisation or well-known cognitive biases [51, 102]. Other examples include applications in negotiation theory [103] and argumentation theory [104], and they have been shown to outperform purely ML-based prediction models.

4 Autonomous Stochastic Multi-Agent Systems (ASMAS)

Contents

The aim of this thesis is to develop a quantitative framework for reasoning about trust. In this chapter, we present our first attempt at devising such a model. We begin by giving a high-level outline of the framework (Section 4.1), followed by a more detailed, formal treatment in Sections 4.2 and 4.3. The example given in Section 4.4 puts theory into practice by modelling a famous money-exchange scenario called trust game using our formalism. Section 4.5 continues the practical theme by presenting algorithms for model checking logical formulae on instances of our model. Finally, we conclude by taking a critical view of the framework, pointing out its limitations and shortcomings (Section 4.6).

Sections 4.1 through 4.4 of this chapter are based on a published journal article [29], coauthored by the writer of this thesis, along with Marta Kwiatkowska and Xiaowei Huang. In particular, major contributions include the running example, formulation of stochastic multiplayer game with cognitive dimension, the induced SMG concept, auxiliary transition function and belief ASMAS construction, with minor contributions in the formulations of the logics, cognitive state and pro-attitude synthesis.

4.1 Overview

Our framework aims to overcome the deficiencies of the approaches described in Section 2.1. It combines a stochastic model derived from concurrent game and a probabilistic logic based on PCTL*. This setting enables us to define cognitive trust in a quantitative manner, as a probabilistic belief, conditioned on agent's mental attitudes. This complex definition of trust is possible thanks to a conceptual division of the model into two dimensions: physical, where agents take actions, and cognitive, representing mental processes that lead to decisions. Inspired by BDI theory [41], each agent is endowed with a set of goals and intentions. Moreover, model checking of trust formulae is considered, and although the general problem is shown undecidable, fragments of the logic are proposed for which tractable model checking algorithms exist.

4.2 Model Description

We now proceed with a formal introduction of the framework. This section describes the model, which is essentially a stochastic multiplayer game with added structure, while Section 4.3 presents a logical language used to reason about properties of systems and express concepts such as trust. Note that our description is by necessity concise – a thorough treatment of the subject, including syntax and semantics of the model and the logic, as well as complexity results, is available in the original publication [29]

An *autonomous stochastic multi-agent system* (ASMAS) is defined by a tuple $(Ags,\ S,\ S_{\text{init}},\ \{Act_A\}_{A\in Ags},\ T,\ L,\ \{O_A\}_{A\in Ags},\ \{\text{obs}_A\}_{A\in Ags},\ \{\Omega_A\}_{A\in Ags},\ \{\lambda_A\}_{A\in Ags},\ \{p_A\}_{A\in Ags})$. More precisely, it is a concurrent game with a finite number of states S, starting in some $s \in S_{\text{init}}$, involving agents Ags taking actions from $Act = \bigtimes_{A\in Ags} Act_A$, which cause the system to move from one state to another according to a stochastic transition function $T : \mathrm{S} \times Act \to \mathcal{D}(\mathrm{S})$. Additionally, partial observability is employed to model agents' inability to fully observe the system's state; it takes the form of a set of observations O_A and an observation function $\text{obs}_A : S \to O_A$ for each $A \in Ags$. Finally, AP is a set of atomic propositions and L is a standard labelling function $L : S \to 2^{AP}$ mapping each state s to a subset of atomic propositions that hold in s. The novel cognitive notions $\{\Omega_A\}_{A\in Ags}$, $\{\lambda_A\}_{A\in Ags}$, $\{p_A\}_{A\in Ags}$, described in detail below, enable one to reason about agents' motivations, beliefs and trust between them.

Cognitive State First of all, in ASMASs, agents are equipped with goals and intentions, where the convention is that goals are (mostly) static mental attitudes, such as politeness or reliability, while intentions are dynamic short-term plans of action, e.g., sharing the money, going shopping or stopping at a red light. The set of possible goals of an agent A is denoted $Goal_A$, while the set of intentions is Int_A. At any given time, an agent may have multiple goals (representing its long-term objectives), but only a single intention (representing its immediate plans). A set of goals and an intention make up what we refer to as agent's cognitive (or mental) state, which is part of a global state of the system and determines agent's behaviour. In fact, in ASMASs, we assume the existence of a one-to-one mapping from intentions to action strategies. Goals, on the other hand, affect agent's behaviour indirectly, by influencing its intentions. The convention that agent's cognitive state determines its behaviour is important enough that we give it a name - we call it the *deterministic behaviour assumption*. In summary, every agent in an ASMAS is equipped with a set of goals and, based on its understanding of the system behaviour and other agents' attitudes, they select an intention (which in turn determines an action strategy) that maximises the probability of achieving those goals.

Cognitive Dimension In order to model agents changing their mental attitudes, in ASMASs, we distinguish two types of transitions: (i) temporal, which represent agents taking actions in the physical space, as in regular multi-agent systems, and (ii) cognitive, representing goal (less common) or intention (more common) changes. Given states s and s' and an agent A, $s \longrightarrow_T^a s'$ denotes a temporal transition via action a, while $s \longrightarrow_C^{A.g.x} s'$ (resp. $s \longrightarrow_C^{A.i.x} s'$) denote a goal (resp. intention) change of agent A. The standard definition of a path as a sequence of states interleaved with temporal transitions is extended in an ASMAS to allow cognitive transitions.

An intuitive way of thinking about an ASMAS is as a collection of stochastic multiplayer games (SMG), each corresponding to a fixed configuration of agents' goals and intentions. A system then starts in one of those SMGs, reflecting the initial mental attitudes of agents, and the execution proceeds by interleaving agents' temporal actions, which preserve the current SMG, and cognitive transitions, which

Table 4.1: Correspondence between game dimensions

Temporal	**Cognitive**
Act_A	$\langle Goal_A, Int_A \rangle$
T	$\{\Omega_A\}_{A \in Ags}$
σ_A	$\pi_A = \langle \pi^g_A, \pi^i_A \rangle$

switch to a different SMG. In fact, as a result of the deterministic behaviour assumption, each individual SMG is fully deterministic and can be viewed as a Markov chain, thereby greatly simplifying computations. The multidimensional visualisation described above gives rise to our division of ASMASs into a *temporal dimension*, consisting of temporal transitions, and a *cognitive dimension*, which comprises all mental changes. We often refer to the SMG that corresponds to a mental configuration of agents as the *induced SMG*.

The conceptual division of ASMASs into temporal and cognitive dimensions serves also as an aid to see the correspondence between the two. For an agent A, its goals $Goal_A$ and intentions Int_A may be viewed as a cognitive counterpart of A's set of actions Act_A.

In the temporal dimension, the transition function T specifies which actions are available at which states. In the cognitive dimension, a similar role is assumed by the *cognitive mechanism* $\Omega_A = \langle \omega^g_A, \omega^i_A \rangle$, which defines legal goal and intention changes of agents, thereby specifying rules of how the system can evolve along the cognitive dimension. Legal goal function $\omega^g_A : S \to \mathcal{P}(\mathcal{P}(Goal_A))$ of agent A is a mapping from states S into sets of subsets of $Goal_A$, where, for any state s, $\omega^g_A(s)$ consists of those subsets of $Goal_A$ that A can take on in state s. Legal intention function is defined analogously.

Finally, we mention that a standard action strategy σ_A of A has its equivalent in the mental dimension, in the form of A's *cognitive strategy* π_A, consisting of a goal strategy $\pi^g_A : \mathrm{FPath}^{\mathcal{M}} \to \mathcal{D}(\mathcal{P}(Goal_A))$ and an intention strategy $\pi^i_A : \mathrm{FPath}^{\mathcal{M}} \to \mathcal{D}(Int_A)$. This correspondence between game dimensions is summarised in Table 4.1.

Partial Observability It should perhaps not be surprising that the notion of partial observability is present in ASMASs. The main motivation for its inclusion is the intrinsic nature of cognitive state, which in general is not observable to others. We capture that concept by introducing a set of observations O_A and an observation function $\mathrm{obs}_A : \mathrm{S} \to O_A$ for each agent A. We say states s and s' are indistinguishable for A iff $\mathrm{obs}_A(s) = \mathrm{obs}_A(s')$. Typically, if s and s' differ only in

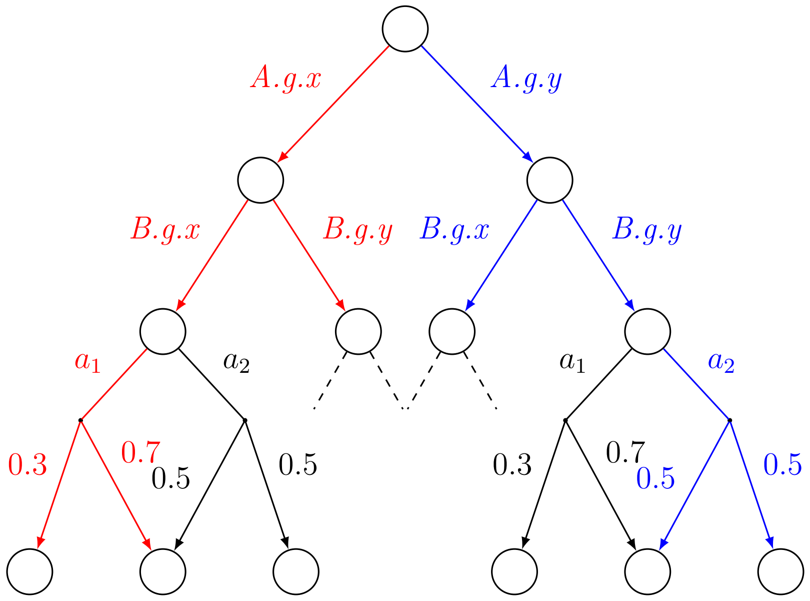


Figure 4.1: Probability spaces as seen by A. All paths with prefixes of a given colour belong to the same probability space. For readability, we only highlight two such, but there are more.

the cognitive state of agent A, say, then $\text{obs}_B(s) = \text{obs}_B(s')$ for all agents B where $B \neq A$. Of course, this might not hold true in certain cases, such as when agent A gives away hints that reveal its intention, for example by turning red when it intends to lie. We mention that the notion of observation extends to paths, with path observation defined as a sequence of observations of states that make up the path.

Transition Type A crucial role in the ASMAS framework is played by the belief functions, which represent each agent's understanding of the current system state and its execution history. Intuitively, based on observations made at each time step of the system's execution, agents maintain a set of possible execution histories. We assume that agents remember both their past observations, as well as the number of states on the path (formally known as *synchronous perfect recall*).

In order to define the belief function, which quantifies likelihoods of possible paths, a probability space on the set of all paths is required. The challenge here is to resolve nondeterminism in the cognitive dimension and still obtain a well-defined probability measure. The difficulties are as follows: (i) agents may change their mental attitudes at arbitrary points, (ii) any agent may make a cognitive transition and (iii) a mental change could take the form of either goal or intention change.

To cope with the above issues, we distinguish five different types of transitions as seen by each agent: a temporal transition, own goal/intention change and other agent's goal/intention change. Formally, given an ASMAS $\mathcal{M}$, a path $\rho = s_0 s_1 s_2 \dots$ and an agent A, $\mathrm{tp}_A(s_k, s_{k+1})$ denotes the type of transition taken to move from s_k to s_{k+1}. The five cases are as follows:

- $\mathrm{tp}_A(s_k, s_{k+1}) = a$ if $s_k \longrightarrow_T^a s_{k+1}$ for some $a \in Act$,
- $\mathrm{tp}_A(s_k, s_{k+1}) = A.g.x$ if $s_k \longrightarrow_C^{A.g.x} s_{k+1}$ for some $x \subseteq \omega_A^g(s_k)$,
- $\mathrm{tp}_A(s_k, s_{k+1}) = A.i.x$ if $s_k \longrightarrow_C^{A.i.x} s_{k+1}$ for some $x \in \omega_A^i(s_k)$,
- $\mathrm{tp}_A(s_k, s_{k+1}) = B.g$ if $s_k \longrightarrow_C^{B.g.x} s_{k+1}$ for another agent $B \in Ags$ and $x \subseteq \omega_B^g(s_k)$,
- $\mathrm{tp}_A(s_k, s_{k+1}) = B.i$ if $s_k \longrightarrow_C^{B.i.x} s_{k+1}$ for another agent $B \in Ags$ and $x \in \omega_B^i(s_k)$.

Given a path ρ, $\mathrm{tp}_A(\rho) = \mathrm{tp}_A(\rho(0), \rho(1)) \cdot \mathrm{tp}_A(\rho(1), \rho(2)) \cdot \dots$ denotes the type of ρ. We assume that each agent can distinguish paths of different types; in particular, it can observe its own mental changes (which should not come as a surprise) and it can tell when other agents make a cognitive transition and whether it's a goal or intention change (but it generally does not know what the new set of goals or new intention is).

Probability Spaces Now, rather than defining one large probability space spanning all the possible paths in an ASMAS, we define one for each type. Since all the paths that an agent cannot distinguish from each other are guaranteed to be in the same probability space, such construction is sufficient for our purposes. As an example, imagine a simple two-agent ASMAS, in which agents start by setting their goals (first A, then B), followed by a global temporal transition, where two actions a_1 and a_2 are available. We use $A.g.x$ to denote a cognitive transition in which A sets its goals to x. We assume that a_1 will be selected if both agents have goals x and that a_2 will be selected if both agents have goals y (in line with deterministic behaviour assumption). We do not consider other goal configurations for simplicity. In such a system, each agent has a unique set of probability spaces. Figure 4.1 illustrates the probability spaces as seen by A; for readability, we only highlight two such (one in red, one in blue), but there are more, corresponding to choosing action a_2 in the left subtree or action a_1 in the right subtree. Figure 4.2 shows how the same system is perceived by B. Note that each of A's cognitive transitions belongs to both probability spaces, depending on which path it is considered part of; hence, their colour is magenta, a mix of red and blue.

Preference Functions What remains to be explained is how nondeterminism in the cognitive dimension is resolved. For example, how does agent A assign probabilities to different goal changes of B? To capture that mechanism, we introduce the concept of *preference functions*, which represent agents' predictions of others' mental attitudes. In particular, for an agent A, its family of preference functions $p_A = \{\langle gp_{A,B}, ip_{A,B}\rangle \mid B \in \text{Ags and } B \neq A\}$ consists of, for each other agent B, a *goal preference function* $gp_{A,B} : \mathrm{S} \to \mathcal{D}(\mathcal{P}(Goal_B))$ and an *intention preference function* $ip_{A,B} : \mathrm{S} \to \mathcal{D}(Int_B)$. These functions associate with each state a probability distribution over possible goal/intention changes of B, as seen by A. In other words, they represent A's beliefs about mental attitudes of B.

With that, based on temporal transition function T and preference functions $\{p_A\}_{A\in Ags}$, we define an auxiliary transition function T_A of an agent $A \in Ags$ as follows for $s, s' \in S$:

$$T_A(s,s') = \begin{cases} T(s,a)(s') & \text{if } \mathrm{tp}_A(s,s') = a, \\ gp_{A,B}(s)(x) & \text{if } \mathrm{tp}_A(s,s') = B.g \text{ and } s \xrightarrow{B.g.x}_C s', \\ ip_{A,B}(s)(x) & \text{if } \mathrm{tp}_A(s,s') = B.i \text{ and } s \xrightarrow{B.i.x}_C s', \\ 1 & \text{if } \mathrm{tp}_A(s,s') = A.g.x \text{ for some } x \in \omega^g_A(s) \\ & \text{or } \mathrm{tp}_A(s,s') = A.i.x \text{ for some } x \in \omega^i_A(s). \end{cases}$$

Hence, T_A resolves the nondeterminism in both the temporal and cognitive dimensions. To define a probability measure, we need one more component: an initial distribution $\mu_0 \in \mathcal{D}(S_{\text{init}})$, representing common prior assumption on initial system state. Then, given an ASMAS $\mathcal{M}$, we define for each agent A a set of probability spaces, one for every path type. Fixing type t, the sample space consists of infinite paths of type t starting in one of the initial states, i.e., $\Omega_{\mathcal{M},t} = \bigcup_{s\in \mathrm{S}, \mu_0(s)>0}\{\delta \in \mathrm{IPath}^{\mathcal{M}}(s) \mid \mathrm{tp}_A(\delta) = t\}$. We associate a probability to each finite path $\rho = s_0...s_n$ consistent with t via function $\mathrm{Pr}_A(\rho) = \mu_0(\rho(0)) \cdot \prod_{0\leq i\leq|\rho|-2} T_A(\rho(i), \rho(i+1))$. We then set $\mathcal{F}_{\mathcal{M}}$ to be the smallest σ-algebra generated by cylinders $\{\mathrm{Cyl}_\rho \cap \Omega_{\mathcal{M},t} \mid \rho \in \mathrm{FPath}^{\mathcal{M}}\}$ and $\mathrm{Pr}^{\mathcal{M}}_A$ to be the unique measure such that $\mathrm{Pr}^{\mathcal{M}}_A(\mathrm{Cyl}_\rho) = \mathrm{Pr}_A(\rho)$. It then follows that $(\Omega_{\mathcal{M},t}, \mathcal{F}_{\mathcal{M}}, \mathrm{Pr}^{\mathcal{M}}_A)$ is a probability space.

Belief Function At any point of system's execution, there are in general multiple possible execution histories that an agent cannot distinguish. Belief function of an agent captures the likelihoods that the agent assigns to those paths. Let $\mathrm{OPath}_A = \{\mathrm{obs}_A(\rho) \mid \rho \in \mathrm{FPath}^{\mathcal{M}}\}$ be the set of all finite observation histories (path observations), and given a path observation $o \in \mathrm{OPath}_A$, let $\mathrm{class}(o) = \{\rho \in$

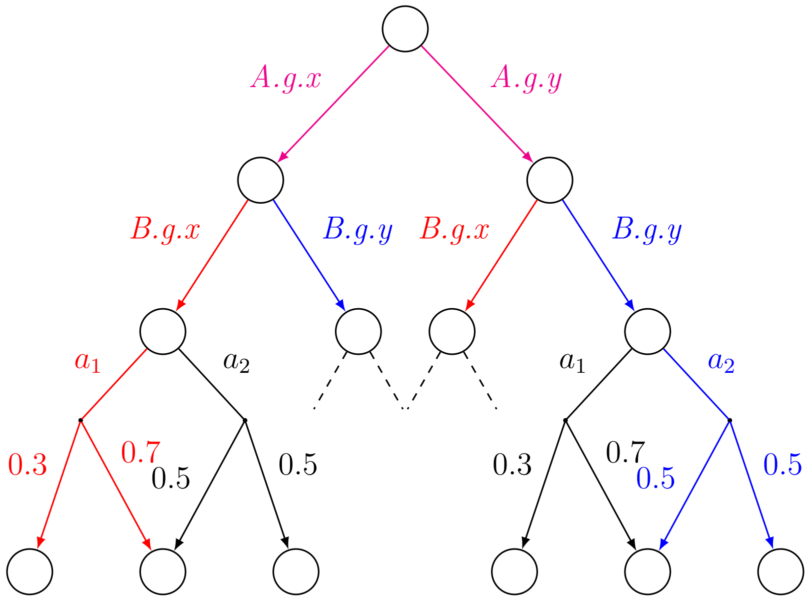


Figure 4.2: Probability spaces as seen by B. For readability, only two are highlighted (red and blue). Note that initial transition is unobservable to B, hence belongs to both probability spaces as part of different paths – this is reflected by its magenta colour.

$\text{FPath}^{\mathcal{M}} \mid \text{obs}_A(\rho) = o\}$ denote the set of all paths that A observes as o. Then the belief function $\texttt{be}_A : \text{OPath}_A \to \mathcal{D}(\text{FPath}^{\mathcal{M}})$ of an agent A is given by:

$$\texttt{be}_A(o)(\rho) = \Pr_A^{\mathcal{M}}(\text{Cyl}_\rho \mid \bigcup_{\rho' \in \text{class}(o)} \text{Cyl}_{\rho'}). \tag{4.1}$$

Hence, given an observation $o \in \text{OPath}_A$, agent A's belief that the current path is $\rho \in \text{class}(o)$ is the probability of ρ conditioned on the fact that the current path belongs to the set $\text{class}(o)$.

4.3 Probabilistic Rational Temporal Logic

We now present the formal language used to express properties of ASMASs. In what follows, let $\mathcal{M} = (Ags, S, S_{\text{init}}, \{Act_A\}_{A \in Ags}, T, L, \{O_A\}_{A \in Ags}, \{\text{obs}_A\}_{A \in Ags}, \{\Omega_A\}_{A \in Ags}, \{\lambda_A\}_{A \in Ags}, \{p_A\}_{A \in Ags})$ be an instance of an ASMAS. The syntax of PRTL* is as follows:

$$\phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid \forall\psi \mid \mathtt{P}^{\bowtie q}\psi \mid \mathbb{G}_A\phi \mid \mathbb{I}_A\phi \mid \mathbb{C}_A\phi \mid \mathbb{B}_A^{\bowtie q}\psi \mid \mathbb{CT}_{A,B}^{\bowtie q}\psi \mid \mathbb{DT}_{A,B}^{\bowtie q}\psi$$
$$\psi ::= \phi \mid \neg\psi \mid \psi \vee \psi \mid \bigcirc\psi \mid \psi\,\mathcal{U}\,\psi$$

where $p \in AP$, $A, B \in Ags$, $\bowtie \in \{<, \leq, >, \geq\}$, and $q \in [0, 1]$.

We first give intuitive interpretations of the novel operators and then define their formal semantics, on which our algorithms are based. Standard PCTL* operators were covered in Section 3.3.2, so we omit discussing them below.

Cognitive Operators $\mathbb{G}_A\phi$, $\mathbb{I}_A\phi$ and $\mathbb{C}_A\phi$ are referred to as cognitive operators and they all express that agent A can ensure future satisfaction of ϕ by modifying its cognitive state in some way. Before we get into the details of each operator, we introduce some notation. Recall that, for any state s, $\omega_A^g(s)$ specifies goal changes that agent A can legally make in a state s. We therefore call them *legal* goal changes. On the other hand, $\pi_A^g(s)$ specifies goal changes that are under A's consideration. We call these *possible* goal changes.

With that, $\mathbb{G}_A\phi$ states that ϕ will hold regardless of A's possible goal changes. In other words, whatever possible goal change A performs, ϕ will hold anyway. Similarly, $\mathbb{I}_A\phi$ expresses that ϕ will hold regardless of A's possible intention changes. Finally, $\mathbb{C}_A\phi$ means that A has a legal intention, taking which will make ϕ true. Formal semantics of those operators is as follows:

- $\mathcal{M}, \rho s \models \mathbb{G}_A\phi$ if $\forall x \in \mathrm{supp}(\pi_A^g(\rho s))\, \exists s' \in \mathrm{S} : s \longrightarrow_C^{A.g.x} s'$ and $\mathcal{M}, \rho s s' \models \phi$,
- $\mathcal{M}, \rho s \models \mathbb{I}_A\phi$ if $\forall x \in \mathrm{supp}(\pi_A^i(\rho s))\, \exists s' \in \mathrm{S} : s \longrightarrow_C^{A.i.x} s'$ and $\mathcal{M}, \rho s s' \models \phi$,
- $\mathcal{M}, \rho s \models \mathbb{C}_A\phi$ if $\exists x \in \omega_A^i(s)\, \exists s' \in \mathrm{S} : s \longrightarrow_C^{A.i.x} s'$ and $\mathcal{M}, \rho s s' \models \phi$.

Note that the semantics is history-dependent, which is why full path ρs is used in the satisfaction relation, rather than state s only, as would be the case for PCTL*. We say that a PRTL* formula, e.g., $\mathbb{G}_A\phi$, is evaluated/verified "at ρ", meaning that it is being evaluated at a point of the system execution when execution history is ρ.

Moreover, we note that, when evaluating PCTL* operators, we assume that agents keep their current mental attitudes, i.e., that the future path is purely temporal. Formally, for an ASMAS $\mathcal{M}$ and execution history ρ, PCTL* operators evaluated at ρ should be interpreted over the stochastic multiplayer game induced from $\mathcal{M}$ with mental state as recorded in the last state of ρ. Furthermore, when evaluating PRTL* formulas, we assume agents can change their goals and intentions at any time. That ensures that the cognitive operators can be applied at any point of execution, as well as meaningfully chained, nested or manipulated in any other way.

Belief and Trust Operators We now turn to the more complex belief and trust operators. Before explaining their semantics, we introduce a concept which is present in the definition of all three of the operators, namely *belief-weighted expectation*. Recall that each agent is equipped with a belief function, which represents its belief about the current execution history of the system. Suppose now that an agent A wishes to evaluate a measurable function $f : \mathrm{FPath}^{\mathcal{M}} \to [0, 1]$ at a given point of execution of the system. Since they do not know what the current path is, the best they can do is to use their belief function to find their subjective expectation of f. Formally, we define the belief-weighted expectation of f as

$$E_{\mathtt{be}_A}[f] = \sum_{\rho \in \mathrm{FPath}^{\mathcal{M}}} \mathtt{be}_A(\rho) \cdot f(\rho).$$

With that, the belief and trust operators are defined as follows.

- $\mathcal{M}, \rho \models \mathbb{B}_A^{\bowtie q}\psi$ if
$$E_{\mathtt{be}_A}[V_{\mathbb{B},\mathcal{M},\psi}] \bowtie q,$$
where the function $V_{\mathbb{B},\mathcal{M},\psi} : \mathrm{FPath}^{\mathcal{M}} \to [0,1]$ is such that
$$V_{\mathbb{B},\mathcal{M},\psi}(\rho') = \mathrm{Pr}_{\mathcal{M},\rho'}(\psi)$$

- $\mathcal{M}, \rho \models \mathbb{CT}_{A,B}^{\bowtie q}\psi$ if
$$E_{\mathtt{be}_A}[V^{\bowtie}_{\mathbb{CT},\mathcal{M},B,\psi}] \bowtie q,$$
where the function $V^{\bowtie}_{\mathbb{CT},\mathcal{M},B,\psi} : \mathrm{FPath}^{\mathcal{M}} \to [0,1]$ is such that
$$V^{\bowtie}_{\mathbb{CT},\mathcal{M},B,\psi}(\rho') = \begin{cases} \sup\limits_{x \in \omega_B^i(last(\rho'))} \mathrm{Pr}_{\mathcal{M},B.i(\rho',x)}(\psi) & \text{if } \bowtie \in \{\geq, >\} \\ \inf\limits_{x \in \omega_B^i(last(\rho'))} \mathrm{Pr}_{\mathcal{M},B.i(\rho',x)}(\psi) & \text{if } \bowtie \in \{<, \leq\}. \end{cases}$$

- $\mathcal{M}, \rho \models \mathbb{DT}_{A,B}^{\bowtie q}\psi$ if
$$E_{\mathtt{be}_A}[V^{\bowtie}_{\mathbb{DT},\mathcal{M},B,\psi}] \bowtie q,$$
where the function $V^{\bowtie}_{\mathbb{DT},\mathcal{M},B,\psi} : \mathrm{FPath}^{\mathcal{M}} \to [0,1]$ is such that
$$V^{\bowtie}_{\mathbb{DT},\mathcal{M},B,\psi}(\rho') = \begin{cases} \inf\limits_{x \in \mathrm{supp}(\pi_B^i(\rho'))} \mathrm{Pr}_{\mathcal{M},B.i(\rho',x)}(\psi) & \text{if } \bowtie \in \{\geq, >\} \\ \sup\limits_{x \in \mathrm{supp}(\pi_B^i(\rho'))} \mathrm{Pr}_{\mathcal{M},B.i(\rho',x)}(\psi) & \text{if } \bowtie \in \{<, \leq\}. \end{cases}$$

Hence, the belief operator $\mathbb{B}_A^{\bowtie q}\psi$ expresses that, from the point of view of agent A, the belief-weighted expectation of the probability of ψ being true is in relation $\bowtie$ with q. Next, the competence trust operator $\mathbb{CT}_{A,B}^{\bowtie q}\psi$ states that A believes that B is capable of implementing ψ (i.e., ensuring probability of ψ being true is in relation $\bowtie$ with q), where capability is understood as existence of a legal intention. Or, more succinctly, A trusts B on its competence to implement ψ. On the other hand, the disposition trust operator $\mathbb{DT}_{A,B}^{\bowtie q}\psi$ expresses that A believes that B is willing to implement ψ, where willingness means that all B's possible intentions guarantee satisfaction of ψ with sufficient probability.

The difference in semantics between competence and disposition trust boils down to two aspects: (i) a trivial one, which is the scope of intentions under consideration (legal vs. possible) and (ii) a more subtle one, which is the way supremum and infimium are used depending on the relational symbol. In particular, when $\bowtie$ is one of $\geq$, $>$, $\mathbb{CT}^{\bowtie q}_{A,B}\psi$ computes the maximum probability of implementing ψ, while $\mathbb{DT}^{\bowtie q}_{A,B}\psi$ is defined as the minimum such probability. In other words, $\mathbb{CT}^{\bowtie q}_{A,B}$ gives an upper bound on B's capability, while $\mathbb{DT}^{\bowtie q}_{A,B}$ gives a lower bound on B's willingness, to implement ψ.

We note that the presented semantics is valid under the deterministic behaviour assumption, in which case the probability of a formula can be determined directly, without referring to action strategies. It is possible to give a more general semantics, with respect to all possible action strategies. However, that adds complexity in the PCTL* portion of the model checking algorithm, but does not affect the verification of novel operators. Therefore, without loss of generality, we focus our attention on the simplified semantics.

Pro-Attitude Synthesis Note that cognitive strategies $\{\pi_A\}_{A\in Ags}$ of agents, despite playing an important role in the framework and in the semantics of logical operators, are not part of the ASMAS definition. That is because they generally lack a finite representation, which is required for all components of the model. A mechanism is therefore introduced that enables one to obtain the infinite structure $\{\pi_A\}_{A\in Ags}$ from a finite structure $\{\lambda_A\}_{A\in Ags}$ called the *guarding mechanism*. The intuitive idea behind λ_A is to specify conditions, expressed in logic, under which agent A takes on a given goal or intention. The assumption is that this resembles reasoning of humans, who do not decide on their plans based on full “execution history”, but rather using their current knowledge or beliefs about the system.

Therefore, a subset of PRTL* is defined, called $\mathcal{L}_A(\text{PRTL}^*)$, which consists of Boolean combinations of atomic propositions and belief and trust operators of agent A. Note that $\mathcal{L}_A(\text{PRTL}^*)$ allows the outermost operator to be of the form $\mathbb{B}^{=?}_{A}\psi$ or $\mathbb{T}^{\bowtie ?}_{A,B}\psi$ (where $\mathbb{T}$ ranges over $\mathbb{CT}$ and $\mathbb{DT}$), i.e., returning the numeric value of A's belief, or a lower/upper bound on its trust towards B. Then, a guarding mechanism λ_A of A is defined as consisting of a *goal guard* function $\lambda^g_A : \mathcal{P}(Goal_A) \to \mathcal{L}_A(\text{PRTL}^*)$ and an *intention guard* function $\lambda^i_A : Int_A \times \mathcal{P}(Goal_A) \to \mathcal{L}_A(\text{PRTL}^*)$. Thus, the guarding mechanism maps goals and intentions to conditions, which can then be evaluated at each point of execution of the system to check whether a given goal/intention is “enabled” or not. Intuitively, an agent's plan of action (intention) depends on their goals and their understanding of the state of the system.

Formally, the synthesis of cognitive strategies from the guarding mechanism is facilitated by the *evaluation functions*; a *goal evaluation function* $eval^g_A$, where for $x \subseteq \mathcal{P}(Goal_A)$, $eval^g_A(x) : \mathrm{FPath}^{\mathcal{M}} \to [0,1]$ is given by

$$eval^g_A(x)(\rho) = \begin{cases} \mathcal{M}, \rho \models \lambda^g_A(x) & \text{if } x \in Goal_A(last(\rho)), \\ 0 & \text{otherwise,} \end{cases}$$

and an *intention evaluation function* $eval^i_A$, such that for any $x \in Int_A$, $y \subseteq \mathcal{P}(Goal_A)$, $eval^i_A(x, y) : \mathrm{FPath}^{\mathcal{M}} \to [0,1]$ is given by

$$eval^i_A(x,y)(\rho) = \begin{cases} \mathcal{M}, \rho \models \lambda^i_A(x,y) & \text{if } x \in Int_A(last(\rho)), \\ 0 & \text{otherwise.} \end{cases}$$

Note that the expressions $\mathcal{M}, \rho \models \mathbb{B}^{=?}_A \psi$ and $\mathcal{M}, \rho \models \mathbb{T}^{\bowtie ?}_{A,B} \psi$ return their corresponding probabilistic values, and $\mathcal{M}, \rho \models \mathbb{B}^{\bowtie q}_A \psi$ and $\mathcal{M}, \rho \models \mathbb{T}^{\bowtie q}_{A,B} \psi$ return the value 0 or 1 depending on the verification result. Finally, using the evaluation functions, cognitive strategy π_A of A is synthesised as

$$\pi^g_A(\rho)(x) = \frac{eval^g_A(x)(\rho)}{\sum_{x \in \omega^g_A(\mathrm{last}(\rho))} eval^g_A(x)(\rho)},$$
$$\pi^i_A(\rho)(x) = \frac{eval^i_A(x,y)(\rho)}{\sum_{x \in \omega^i_A(\mathrm{last}(\rho))} eval^i_A(x,y)(\rho)}.$$

4.4 Trust Game Example

To put the ideas introduced above into practice, we use ASMAS to model a well-known scenario called trust game. Introduced by Berg et al. [30] more than twenty years ago, this experimental setup has been used numerous times to study trust through economic interactions.

In this chapter, we consider a simplified version of the example, referred to as basic trust game [105]. It involves two agents, Alice and Bob, entering a following money-investing scenario. They start with \$10 and \$5, respectively. Then, Alice may invest her \$10 with Bob, or withhold the money. In the former case, Bob turns the \$15 into \$40, which he can then share (equally) with Alice, or keep all for himself.

This simple interaction is commonly modelled as a stochastic multiplayer game (SMG) $\mathcal{T}$, with a set of agents $\mathrm{Ags} = \{\mathrm{Alice}, \mathrm{Bob}\}$, with two actions each $\mathrm{Act}_{\mathrm{Alice}} = \{withhold, invest\}$, $\mathrm{Act}_{\mathrm{Bob}} = \{keep, share\}$. States and transitions are as in Figure 4.3. Monetary exchanges are captured by a reward structure $r : \mathrm{S} \times \mathrm{Act} \to \mathbb{R}$ consistent with payoffs associated to terminal nodes in Figure 4.3.

Standard analysis based on Nash equilibria reveals that it is optimal for Alice to withhold her money and for Bob to keep his profits (in case Alice invests,

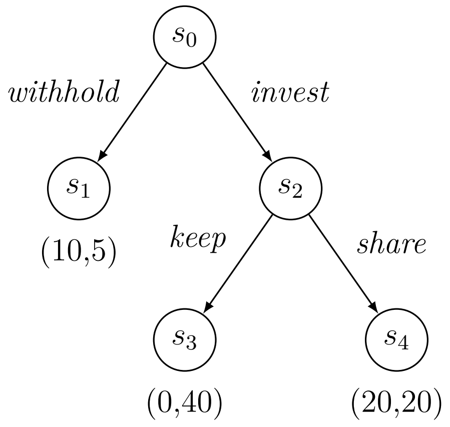


Figure 4.3: Simple trust game

Table 4.2: Payoffs of a simple trust game with trust as a decision factor

$(Alice, Bob)$	Share	Keep
Invest	(20,20+5)	(0,40-20)
Withhold	(10,5)	(10,5)

which she won't, the theory goes...). Importantly, this conclusion is reached by assuming agents are solely motivated by money and they are perfectly rational. Arguably, the prediction is somewhat unsettling with regards to human nature and the forecast outcome clearly undesirable in terms of social welfare. Not all hope is lost, however, as humans have been shown to share the money more often than the theory predicts [54]. This has led some authors to put forward alternative formulations of the example, or new theoretical models aiming to explain agents' behaviour. In one such attempt, payoffs of the game have been modified (see Table 4.2) to reflect the changing value of trust resulting from different actions taken by agents [105]. With updated utilities, the new Nash equilibrium dictates Alice invests her money with Bob, who then shares his profits with Alice. This works well for this particular example, but a more general approach is needed to cover a wider range of scenarios. This is where ASMAS comes in.

We start with a basic model $\mathcal{T}$ defined above and extend it with a cognitive dimension as follows. Recall that each agent in an ASMAS needs to be endowed with goals and intentions. The framework does not formally enforce any structure or constraints for goals and intentions, but an established convention dictates that goals represent static personality traits, while intentions describe what agents do, and can be identified with action strategies. In this case, we assign goals and intentions as follows: $Goal_{Alice} = \{passive, active\}$, $Int_{Alice} = \{passive, active\}$, $Goal_{Bob} =$

Table 4.3: Strategies for Alice and Bob

Strategy	*withhold*	*invest*	*keep*	*share*
$\sigma_{passive}$	0.7	0.3		
σ_{active}	0.1	0.9		
σ_{share}			0.0	1.0
σ_{keep}			1.0	0.0

$\{\mathit{investor}, \mathit{opportunist}\}$, $\mathit{Int}_{\mathit{Bob}} = \{\mathit{share}, \mathit{keep}\}$. Note that Alice's intentions mirror her goals; this stems from our (simplifying) assumption that her intentions are solely determined by her goals. We also define action strategies corresponding to agents' intentions, summarised in Table 4.3.

The game is then augmented with a cognitive dimension, resulting in an extended system depicted in Figure 4.4. We follow a convention where goals are set initially and don't change throughout the execution, while an agent determines their intention before every action, which represents the process of decision making. However, recall that agents may take cognitive transitions at any time; the figure shows one possible interleaving. Note also that our assumption that Alice's goals fully determine her intention allows us to omit her intention-setting transitions. Finally, we point out that physical transitions are annotated with probabilities, reflecting the deterministic behaviour assumption which postulates a one-to-one mapping of intentions to strategies.

Cognitive state of agents is part of the game's state, which is a tuple

$$(a_{\mathit{Alice}}, a_{\mathit{Bob}}, gs_{\mathit{Alice}}, gs_{\mathit{Bob}}, is_{\mathit{Alice}}, is_{\mathit{Bob}}),$$

where $a_{\mathit{Alice}} \in \mathit{Act}_{\mathit{Alice}} \cup \{\perp\}$ is Alice's last action (or $\perp$ representing "silent" action, i.e., lack of action), $a_{\mathit{Bob}} \in \mathit{Act}_{\mathit{Bob}} \cup \{\perp\}$ is Bob's last action, $gs_{\mathit{Alice}} \subseteq \mathit{Goal}_{\mathit{Alice}} \cup \{\perp\}$ is Alice's set of goals (similarly for Bob) and $is_{\mathit{Alice}} \in \mathit{Int}_{\mathit{Alice}} \cup \{\perp\}$ is Alice's intention (similarly for Bob). For example, $s_4 = (\perp, \perp, \mathit{passive}, \mathit{opportunist}, \mathit{passive}, \perp)$ and $s_{13} = (\mathit{withhold}, \perp, \mathit{active}, \mathit{opportunist}, \mathit{active}, \perp)$.

Next, we give the guarding mechanism, which, when combined with pro-attitude synthesis, yields intention strategies for agents. Note that goal strategies are not required here as we assume goals are static. For Alice, the task is trivial, since

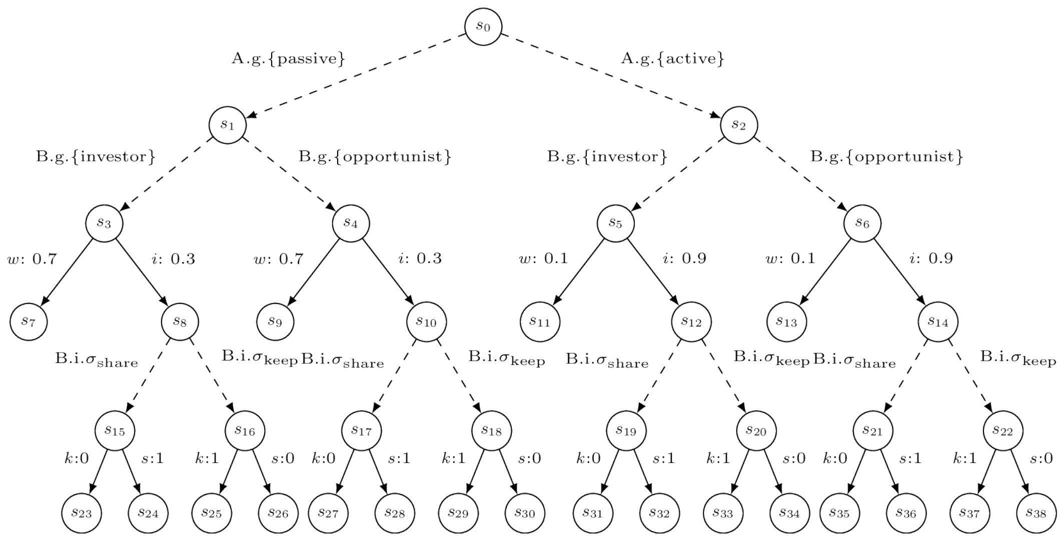


Figure 4.4: Trust game with cognitive dimension

her intention is determined by her goals.

$$\begin{aligned}\lambda^i_{Alice}(active, \{active\}) &= \top,\\ \lambda^i_{Alice}(active, \{passive\}) &= \bot,\\ \lambda^i_{Alice}(passive, \{active\}) &= \bot,\\ \lambda^i_{Alice}(passive, \{passive\}) &= \top.\end{aligned}$$

For Bob, the intention guard takes the following form:

$$\begin{aligned}\lambda^i_{Bob}(share, \{investor\}) &= \mathbb{B}^{>0.7}_{Bob}\, active_{Alice},\\ \lambda^i_{Bob}(keep, \{investor\}) &= \neg\mathbb{B}^{>0.7}_{Bob}\, active_{Alice},\\ \lambda^i_{Bob}(share, \{opportunist\}) &= \bot,\\ \lambda^i_{Bob}(keep, \{opportunist\}) &= \top.\end{aligned} \tag{4.2}$$

Put another way, when Bob is an opportunist, he keeps the money for himself, but when he is an investor, he shares the money if his belief in Alice being active is sufficiently high.

For the agents to reason about the behaviour of their opponent, they need to quantify the likelihood of adversary's mental attitudes. This is represented in the framework by preference functions. For example, Alice's goal preference function over Bob may be

$$gp_{Alice,Bob}(s_k) = \langle investor \mapsto 1/2, opportunist \mapsto 1/2\rangle \quad \text{for } k \in \{1, 2\}, \tag{4.3}$$

i.e., a uniform prior on Bob's goal, which represents lack of knowledge. Her intention preference function could be

$$\begin{aligned} ip_{Alice,Bob}(s_k) &= \langle share \mapsto 3/4, keep \mapsto 1/4\rangle \quad \text{for } k \in \{8,12\}, \\ ip_{Alice,Bob}(s_k) &= \langle share \mapsto 0, keep \mapsto 1\rangle \quad \text{for } k \in \{10,14\}. \end{aligned} \tag{4.4}$$

Note that the preference function as given above should be thought of as an initial preference function. If more iterations of a given system are modelled, agents' preferences are updated, reflecting new information gained by agents. We postpone the discussion of preference function update until later. For now, we give Bob's goal preference function:

$$gp_{Bob,Alice}(s_0) = \langle passive \mapsto 1/3, active \mapsto 2/3\rangle. \tag{4.5}$$

We also assume for simplicity that Bob knows that Alice's intention is fully determined by her goal, so he doesn't need an intention preference function.

With that setup, beliefs of agents and, more importantly, trust values may be computed. The framework supports evaluation of various logical formulae, the most interesting ones being the belief, competence trust and disposition trust operators.

Let's first consider competence trust operator. For example, we may be interested in computing how much Alice trusts Bob's competence to share the profit with her, or how much Bob trusts Alice's competence to invest her money with him. Since both tasks require no skill, we expect the value of trust to be high in both cases. We therefore consider a formula such as

$$\mathbb{CT}^{\bowtie ?}_{Bob,Alice} \bigcirc (a_{Alice} = invest),$$

(with $\bowtie$ one of $\geq$ or $\leq$) evaluated in state s_3, say (recall that $\bigcirc$ is the next state operator). Note that we use an expression $(a_{Alice} = invest)$ as a shorthand for an atomic proposition that holds true in states that satisfy that identity. We first note that Alice has two legal intentions (we assume all possible intention changes are legal for agents at all times) in state s_3 – active and passive. Under active, the probability of her investing is 0.9; under passive, it is 0.3. Note also that state s_5 is indistinguishable from s_3 for Bob, but Alice's legal intentions are the same in both states, so, letting $\psi = \bigcirc(a_{Alice} = invest)$, $\rho_1 = s_0s_1s_3$ and $\rho_2 = s_0s_2s_5$, the value of $V^{\bowtie}_{\mathbb{CT},\mathcal{M},Bob,\psi}(\rho)$ is the same for $\rho = \rho_1$ and $\rho = \rho_2$. In particular,

$$\begin{aligned} V^{\geq}_{\mathbb{CT},\mathcal{M},\text{Bob},\psi}(\rho) &= 0.9, \\ V^{\leq}_{\mathbb{CT},\mathcal{M},\text{Bob},\psi}(\rho) &= 0.3, \end{aligned}$$

where we used $V^{\bowtie}_{\mathbb{CT},\mathcal{M},\mathrm{Bob},\psi}(\rho)$ to denote the common value of $V^{\bowtie}_{\mathbb{CT},\mathcal{M},\mathrm{Bob},\psi}(\rho_1)$ and $V^{\bowtie}_{\mathbb{CT},\mathcal{M},\mathrm{Bob},\psi}(\rho_2)$. Therefore,

$$\mathcal{M}, \rho \models \mathbb{CT}^{\geq 0.9}_{\mathrm{Bob},\mathrm{Alice}} \bigcirc (a_{\mathrm{Alice}} = \mathit{invest})$$
$$\mathcal{M}, \rho \models \mathbb{CT}^{\leq 0.3}_{\mathrm{Bob},\mathrm{Alice}} \bigcirc (a_{\mathrm{Alice}} = \mathit{invest})$$

Similar reasoning leads us to compute Alice's trust in Bob's competence to share his profit (letting $\rho = s_0 s_1 s_3 s_8$):

$$\mathcal{M}, \rho \models \mathbb{CT}^{\geq 1}_{\mathrm{Alice},\mathrm{Bob}} \bigcirc (a_{\mathrm{Bob}} = \mathit{share}),$$
$$\mathcal{M}, \rho \models \mathbb{CT}^{\leq 0}_{\mathrm{Alice},\mathrm{Bob}} \bigcirc (a_{\mathrm{Bob}} = \mathit{share}).$$

Bob's action strategies are pure, resulting in competence trust values of 0 and 1. Overall, as expected, the bounds on agents' competence trust are quite wide, reflecting the fact that actions in the trust game require no skill.

In this case, disposition trust is the more relevant of the two trust operators. We now repeat the above computations, but with $\mathbb{CT}$ operator swapped for $\mathbb{DT}$. We note that the main difference in the evaluation of disposition trust, compared to competence trust, is that the former is formulated in terms of (the support of the) cognitive strategy of the trustee. Therefore, to evaluate $\mathbb{DT}^{\bowtie ?}_{\mathrm{Bob},\mathrm{Alice}}\psi$, we need to synthesise Alice's intention strategy. Fortunately, this is an easy task, since Alice's intention is determined by her goal. Therefore, for ρ being any path of length 2,

$$\begin{aligned} \mathit{eval}^i_{\mathrm{Alice}}(\mathit{active}, \{\mathit{active}\})(\rho) &= 1, \\ \mathit{eval}^i_{\mathrm{Alice}}(\mathit{passive}, \{\mathit{active}\})(\rho) &= 0, \\ \mathit{eval}^i_{\mathrm{Alice}}(\mathit{active}, \{\mathit{passive}\})(\rho) &= 1, \\ \mathit{eval}^i_{\mathrm{Alice}}(\mathit{passive}, \{\mathit{passive}\})(\rho) &= 0, \end{aligned}$$

and so

$$\pi^i_{\mathrm{Alice}}(\rho_1) = \langle \mathit{active} \mapsto 1, \mathit{passive} \mapsto 0 \rangle \quad \text{for } \rho_1 \in \{s_0 s_2 s_5, s_0 s_2 s_6\},$$
$$\pi^i_{\mathrm{Alice}}(\rho_2) = \langle \mathit{active} \mapsto 0, \mathit{passive} \mapsto 1 \rangle \quad \text{for } \rho_2 \in \{s_0 s_1 s_3, s_0 s_1 s_4\}.$$

With that, the V-functions are evaluated as follows:

$$\begin{aligned} V^{\bowtie}_{\mathbb{DT},\mathcal{M},\mathrm{Alice},\psi}(s_0 s_1 s_3) &= 0.3, \\ V^{\bowtie}_{\mathbb{DT},\mathcal{M},\mathrm{Alice},\psi}(s_0 s_1 s_4) &= 0.3, \\ V^{\bowtie}_{\mathbb{DT},\mathcal{M},\mathrm{Alice},\psi}(s_0 s_2 s_5) &= 0.9, \\ V^{\bowtie}_{\mathbb{DT},\mathcal{M},\mathrm{Alice},\psi}(s_0 s_2 s_6) &= 0.9, \end{aligned}$$

where $\bowtie$ is any of $>, \geq, <, \leq$.

Bob's belief is computed easily from his preference function, in particular, letting $o_1 = \mathrm{obs}_{\mathrm{Bob}}(s_0 s_1 s_3) = \mathrm{obs}_{\mathrm{Bob}}(s_0 s_2 s_5)$ and $o_2 = \mathrm{obs}_{\mathrm{Bob}}(s_0 s_1 s_4) = \mathrm{obs}_{\mathrm{Bob}}(s_0 s_2 s_6)$,

$$\begin{aligned}
\mathtt{be}_{Bob}(o_1, s_0 s_1 s_3) &= 1/3, \\
\mathtt{be}_{Bob}(o_1, s_0 s_2 s_5) &= 2/3, \\
\mathtt{be}_{Bob}(o_2, s_0 s_1 s_4) &= 1/3, \\
\mathtt{be}_{Bob}(o_2, s_0 s_2 s_6) &= 2/3.
\end{aligned}$$

Therefore, for ρ being any path of length 2,

$$\mathcal{M}, \rho \models \mathbb{DT}^{\bowtie 0.7}_{\mathrm{Bob},\mathrm{Alice}} \bigcirc (a_{\mathrm{Alice}} = \mathit{invest}) \text{ for } \bowtie \in \{\leq, \geq\}.$$

Because Alice's intention strategy is pure, upper and lower bounds on Bob's disposition trust agree. In this case, a single value of trust may be given.

Let's now see how Alice's trust in Bob compares to the above. We consider the formula $\mathbb{DT}^{\bowtie ?}_{\mathrm{Alice},\mathrm{Bob}} \bigcirc (a_{\mathrm{Bob}} = \mathit{share})$ evaluated at paths $\rho_1 = s_0 s_1 s_3 s_8$ and $\rho_3 = s_0 s_2 s_5 s_{12}$. Let us also denote path $s_0 s_1 s_4 s_{10}$ by ρ_2 and path $s_0 s_2 s_6 s_{14}$ by ρ_4. We start by synthesizing Bob's intention strategy. Recall that Bob's intention when he is an investor is guarded by an expression $\mathbb{B}^{>0.7}_{Bob} \mathit{active}_{Alice}$. Hence, we must first compute Bob's belief. Letting $o_1 = \mathrm{obs}_{\mathrm{Bob}}(\rho_1) = \mathrm{obs}_{\mathrm{Bob}}(\rho_3)$ and $o_2 = \mathrm{obs}_{\mathrm{Bob}}(\rho_2) = \mathrm{obs}_{\mathrm{Bob}}(\rho_4)$, and using Bob's goal preference function and the definition of belief (Equation 4.1), we compute

$$\begin{aligned}
\mathtt{be}_{\mathrm{Bob}}(o_1, \rho_1) &= 1/7, \\
\mathtt{be}_{\mathrm{Bob}}(o_1, \rho_3) &= 6/7, \\
\mathtt{be}_{\mathrm{Bob}}(o_2, \rho_2) &= 1/7, \\
\mathtt{be}_{\mathrm{Bob}}(o_2, \rho_4) &= 6/7.
\end{aligned}$$

Therefore, for ρ being any of $\rho_1, \rho_2, \rho_3, \rho_4$,

$$\mathcal{M}, \rho \models \mathbb{B}^{>0.7}_{\mathrm{Bob}} \mathit{active}_{\mathrm{Alice}},$$

and so

$$\begin{aligned}
\mathit{eval}^i_{\mathrm{Bob}}(\mathit{share}, \{\mathit{investor}\})(\rho) &= 1, \\
\mathit{eval}^i_{\mathrm{Bob}}(\mathit{keep}, \{\mathit{investor}\})(\rho) &= 0, \\
\mathit{eval}^i_{\mathrm{Bob}}(\mathit{share}, \{\mathit{opportunist}\})(\rho) &= 0, \\
\mathit{eval}^i_{\mathrm{Bob}}(\mathit{keep}, \{\mathit{opportunist}\})(\rho) &= 1.
\end{aligned}$$

Hence, Bob's synthesised intention strategy is as follows:

$$\pi^i_{\text{Bob}}(\rho) = \langle \mathit{share} \mapsto 1, \mathit{keep} \mapsto 0\rangle \quad \text{for } \rho \in \{\rho_1, \rho_3\},$$
$$\pi^i_{\text{Bob}}(\rho) = \langle \mathit{share} \mapsto 0, \mathit{keep} \mapsto 1\rangle \quad \text{for } \rho \in \{\rho_2, \rho_4\}.$$

With that, we compute the V-function as follows, letting $\psi = \bigcirc(a_{\text{Alice}} = \mathit{invest})$:

$$V^{\bowtie}_{\mathbb{DT},\mathcal{M},\text{Bob},\psi}(\rho) = 1 \text{ for } \rho \in \{\rho_1, \rho_3\} \text{ and } \bowtie \in \{\geq, \leq\},$$
$$V^{\bowtie}_{\mathbb{DT},\mathcal{M},\text{Bob},\psi}(\rho) = 0 \text{ for } \rho \in \{\rho_2, \rho_4\} \text{ and } \bowtie \in \{\geq, \leq\}.$$

Now, recalling Alice's uniform goal preference function and, arising from it, uniform belief, her disposition trust can be computed as

$$\mathcal{M}, \rho \models \mathbb{DT}^{\bowtie 0.5}_{\text{Alice},\text{Bob}} \bigcirc (a_{\text{Bob}} = \mathit{share}) \text{ for } \bowtie \in \{\leq, \geq\},$$

where ρ is any of ρ_1, ρ_2, ρ_3, ρ_4. Again, upper and lower bounds on the value of trust agree, due to the intention strategy being pure. Alice's trust in Bob sharing the money upon her investment is equal to 0.5, reflecting her uncertainty of Bob's mental attitudes.

To wrap up the example, we consider an alternative mental approach of Bob, characterised by the following intention guards:

$$\lambda^i_{Bob}(\mathit{share}, \{\mathit{investor}\}) = \mathbb{B}^{=?}_{Bob}\mathit{active}_{Alice},$$
$$\lambda^i_{Bob}(\mathit{keep}, \{\mathit{investor}\}) = \mathbb{B}^{=?}_{Bob}\neg\mathit{active}_{Alice},$$
$$\lambda^i_{Bob}(\mathit{share}, \{\mathit{opportunist}\}) = \bot,$$
$$\lambda^i_{Bob}(\mathit{keep}, \{\mathit{opportunist}\}) = \top.$$

This is very similar to the previous setup, except that now guards are quantitative, rather than qualitative, and take the value of Bob's belief. The updated evaluation functions are

$$\mathit{eval}^i_{\text{Bob}}(\mathit{share}, \{\mathit{investor}\})(\rho) = 6/7,$$
$$\mathit{eval}^i_{\text{Bob}}(\mathit{keep}, \{\mathit{investor}\})(\rho) = 1/7,$$
$$\mathit{eval}^i_{\text{Bob}}(\mathit{share}, \{\mathit{opportunist}\})(\rho) = 0,$$
$$\mathit{eval}^i_{\text{Bob}}(\mathit{keep}, \{\mathit{opportunist}\})(\rho) = 1,$$

which leads to slightly different intention strategy for Bob:

$$\pi^i_{\text{Bob}}(\rho) = \langle \mathit{share} \mapsto 6/7, \mathit{keep} \mapsto 1/7\rangle \quad \text{for } \rho \in \{\rho_1, \rho_3\},$$
$$\pi^i_{\text{Bob}}(\rho) = \langle \mathit{share} \mapsto 0, \mathit{keep} \mapsto 1\rangle \quad \text{for } \rho \in \{\rho_2, \rho_4\}.$$

The important difference is that the support of Bob's intention strategy when he is an investor is larger now, giving rise to different trust evaluation. In particular, the V-functions are now

$$\begin{aligned}
V^{\geq}_{\mathbb{DT},\mathcal{M},\text{Bob},\psi}(\rho) &= 0 \text{ for } \rho \in \{\rho_1, \rho_3\},\\
V^{\leq}_{\mathbb{DT},\mathcal{M},\text{Bob},\psi}(\rho) &= 1 \text{ for } \rho \in \{\rho_1, \rho_3\},\\
V^{\bowtie}_{\mathbb{DT},\mathcal{M},\text{Bob},\psi}(\rho) &= 0 \text{ for } \rho \in \{\rho_2, \rho_4\}.
\end{aligned}$$

With that, for $\rho \in \{\rho_1, \rho_2, \rho_3, \rho_4\}$,

$$\begin{aligned}
\mathcal{M}, \rho &\models \mathbb{DT}^{\geq 0}_{\text{Alice},\text{Bob}} \bigcirc (a_{\text{Bob}} = \textit{share}),\\
\mathcal{M}, \rho &\models \mathbb{DT}^{\leq 1/2}_{\text{Alice},\text{Bob}} \bigcirc (a_{\text{Bob}} = \textit{share}).
\end{aligned}$$

Hence, a small change in Bob's mental reasoning mechanism results in a significant change in the bounds on Alice's trust provided by the framework. In fact, it is easy to see that if Bob's intention strategy when he is an opportunist was also mixed, the upper bound on disposition trust would be 1, effectively rendering the bounds useless. This suggests that in order to ensure dispositional trust operator provides tight bounds, intention strategies should ideally be kept pure, or at least have small supports.

4.5 Model Checking Algorithms

The general verification procedure for ASMASs starts by performing pro-attitude synthesis to obtain cognitive strategies of agents, followed by model checking a given formula ϕ on the updated system. Unfortunately, both the synthesis and the model checking have been shown to be undecidable [29]. However, to counter the undecidability results, three fragments of PRTL* are proposed for which a model checking algorithm can be formulated. In what follows, we work with one such subset, called BPRTL*, and present an algorithm for model checking BPRTL* formulae against an instance of an ASMAS. It is based on the PCTL* model checking algorithm presented in Section 3.3.3. However, the standard approach must be adapted to deal with intricacies of ASMASs and to handle the new operators PRTL* introduces.

4.5.1 Challenges

Below, we list the aspects of the framework that necessitate modification to standard methods.

Undecidability Intuitively, undecidability of PRTL* model checking stems from the history-dependence of PRTL* semantics – to check whether a given belief or trust formula eventually holds on a given path, an infinite number of checks might have to be performed. Therefore, to ensure decidability, we work with a bounded fragment of PRTL*, called BPRTL*, which replaces unbounded until operator with a step-bounded one. As a result, one only needs to check a finite prefix of an infinite path to decide whether it satisfies a given BPRTL* formula.

History Dependence Another consequence of the history dependence of PRTL* semantics is that the usual model checking approach of computing satisfaction sets of formulae cannot be employed. Instead, our algorithm starts with a given path and proceeds recursively in a top-down fashion. An important complication is the way path formulae are handled. Recall from Section 3.3.3 that the standard approach involves computing satisfaction sets of maximal state subformulae, allowing one to treat each path formula as an LTL formula and use automata-based techniques. However, due to history dependence, this method has to be altered.

Instead, we use boundedness of the BPRTL* fragment, which means that every path formula can be shown to hold, or not, on an infinite path δ by inspecting its finite prefixes. This enables us to define a function which takes (i) a finite execution history ρ that has s as its last state and (ii) a path formula ψ as arguments, and returns finite path fragments starting in s such that infinite paths starting in s and satisfying ψ are precisely those that have one of the returned fragments as a prefix. This recursive procedure is formally specified as Algorithm 4. It involves a traversal of the parse tree of ψ, with execution history ρ appropriately updated for every subformula being processed.

The procedure makes use of auxiliary functions *complement*, *union* and *intersection* that manipulate sets of prefixes. For example, *complement* accepts a set of prefixes starting in some state s and returns another set of prefixes starting in s such that every infinite path starting in s has as prefix an element of exactly one of the sets. The complement of an input set of prefixes S is computed by a procedure that maintains a list of prefixes l, initialised to consist of a trivial prefix s, and processing them in order until no prefixes remain. At each step, first prefix in the list, call it ρ_0, is removed and there are three possibilities: (i) if ρ_0 is equal to some prefix in the input set S, continue to the next prefix in l; (ii) if ρ_0 is a proper prefix of some other prefix in the input set, add all the one-step extensions of ρ_0 to l; and (iii) otherwise, add ρ_0 to the output set.

Next, *intersection* operates on two input sets of prefixes, say A and B, and returns a set of prefixes such that every infinite path generated by the resulting set has a prefix in A and a prefix in B. The way this intersection of sets of prefixes is computed is by iterating over both input sets, in turn, and adding a prefix ρ_1 from one input set to the output set if there is a prefix ρ_2 in the other output set such that ρ_2 is a prefix of ρ_1. Finally, *union* of two sets of prefixes is computed by taking the union of the sets and simplifying the result. The latter task is captured by the function *simplify* which iterates over prefixes in the input set and, for each, checks whether it can be replaced by a shorter prefix. Intuitively, this is the case if the input set contains all the possible one-step extensions of the shorter prefix.

Partial Observability The next aspect which requires our close attention is partial observability. It is known that strategy synthesis and many other decision problems for POMDPs are undecidable [106]. However, in autonomous stochastic multi-agent systems, strategies, which are identified with intentions, are given as part of the system definition. It is important that those strategies are consistent with partial observability, i.e., that they specify the same distributions on paths an agent cannot distinguish. Nonetheless, when evaluating the formulae, we assume the future execution of the system is deterministic, and therefore, partial observability does not introduce any difficulties. It only comes into play when computing belief-weighted expectation as part of model checking belief and trust formulae (see Section 4.5.2). Note that the presented model checking algorithms operate on sequences of states, rather than sequences of observations, even though agents might not be able to distinguish them from other state sequences with the same observation. This stems from the fact that we don't make any assumptions regarding the truth values of temporal formulae on indistinguishable paths. In other words, we could have an atomic proposition that holds on one path and does not hold on another, even though both paths have the same observation for a given agent. If we replaced state sequences with observation sequences, such atomic proposition could not be evaluated meaningfully. However, belief and trust formulae, which intuitively represent agent's subjective view of system's execution, are consistent with agent's own partial observability and give the same result regardless on which indistinguishable path they are evaluated (details in Section 4.5.2).

Algorithm 4: Computing prefixes of paths satisfying a given path formula

```
input  : ASMAS $\mathcal{M}$, an execution history $\rho$ in $\mathcal{M}$, path formula $\psi$
output : set of prefixes of paths satisfying a given path formula
function findSatisfyingPrefixes($\mathcal{M}$, $\rho$, $\psi$):
    $s \leftarrow last(\rho)$;
    switch $\psi$ do
        case $\phi$ do
            /* procedure modelCheck is given below */
            return $modelCheck(\mathcal{M}, \rho, \phi)$ ? $\{s\}$ : $\emptyset$;
        end
        case $\neg\psi'$ do
            return $complement(findSatisfyingPrefixes(\mathcal{M}, \rho, \psi'))$;
        end
        case $\psi_1 \vee \psi_2$ do
            return $union(findSatisfyingPrefixes(\mathcal{M}, \rho, \psi_1),$
             $findSatisfyingPrefixes(\mathcal{M}, \rho, \psi_2))$;
        end
        case $\bigcirc\psi'$ do
            $result \leftarrow \emptyset$;
            /* post(s) is the set of successors of s */
            foreach $s' \in$ post(s) do
                foreach prefix $\in$ findSatisfyingPrefixes$(\mathcal{M}, \rho s', \psi')$ do
                    $result \leftarrow result \cup concat(s, prefix)$
                end
            end
            return $result$;
        end
        case $\psi_1 \mathcal{U}^{\leq k} \psi_2$ do
            $result \leftarrow findSatisfyingPrefixes(\mathcal{M}, \rho, \psi_2)$;
            if $k = 0$ then
                return $simplify(result)$;
            end
            $ps_1 \leftarrow findSatisfyingPrefixes(\mathcal{M}, \rho, \psi_1)$;
            foreach $s' \in post(s)$ do
                $prefixes \leftarrow findSatisfyingPrefixes(\mathcal{M}, \rho s', \psi_1 \mathcal{U}^{\leq k-1} \psi_2)$;
                foreach prefix $\in$ prefixes do
                    $prefix \leftarrow concat(s, prefix)$;
                end
                $result \leftarrow result \cup intersection(ps_1, prefixes)$;
            end
            return $simplify(result)$;
        end
    end
end
```

Pro-Attitude Synthesis Most of the novel PRTL* operators are defined in terms of cognitive strategies of agents, which are not part of the ASMAS definition. However, in the algorithms described below, we take agents' cognitive strategies as given, since they are easy to compute using pro-attitude synthesis. In particular, to get an agent's intention strategy at a given point of execution, it suffices to iterate over its legal intention changes and compute the weight of each using the evaluation function for that agent. Given that the language used to express the guarding mechanism is a subset of BPRTL*, the weights are computed using model checking techniques presented below. Then the probability of each intention change is its weight divided by the total weight of all intention changes. The computed distribution may be stored for future use.

The above assumes that a cognitive strategy of an agent for a given execution history is computed only when required by the model checking algorithm. However, for the sake of efficiency, a tool implementing the presented algorithms might start computing cognitive strategies of all agents in advance, as soon as the model is compiled. A separate background thread could be assigned specifically for that purpose. The computation would proceed according to the length of the execution history, i.e., starting by computing action distributions corresponding to the shortest paths, and continuing on to longer ones. However, alternative heuristic is possible and could be deployed if shown to be superior. We expect that most formulae of the bounded fragment only need checking for a relatively short path, and therefore we anticipate significant time savings by employing opportunistic ahead-of-time computation of cognitive strategies.

4.5.2 The Algorithms

Hereinafter, we take $\mathcal{M} = (Ags, S, S_{\text{init}}, \{Act_A\}_{A\in Ags}, T, L, \{O_A\}_{A\in Ags}, \{\text{obs}_A\}_{A\in Ags}, \{\Omega_A\}_{A\in Ags}, \{\lambda_A\}_{A\in Ags}, \{p_A\}_{A\in Ags})$ to be an instance of an ASMAS, ρ a finite path in $\mathcal{M}$ and $A \in Ags$. We want to model check a PRTL* formula ϕ_0, i.e. decide whether $\mathcal{M}, \rho \models \phi_0$. The procedure has two possible outcomes, *true* and *false*, indicating that $\mathcal{M}, \rho \models \phi_0$ and $\mathcal{M}, \rho \not\models \phi_0$, respectively.

The main model checking procedure is recursive in nature (Algorithm 5) and specified on a case-by-case basis, in terms of the principal connective of ϕ_0. Propositional operators are straightforward to handle. For the universal quantification and probability operator, we utilise *findSatisfyingPrefixes*. Since the set of prefixes it returns is minimal, $\forall\psi$ holds if and only if the last state of ρ, last(ρ), is the sole prefix returned. It is also easy to compute the total probability of paths satisfying ψ by summing up the probabilities of cylinder sets corresponding

Algorithm 5: PRTL* model checking algorithm

```
input  : ASMAS $\mathcal{M}$, a path $\rho$ in $\mathcal{M}$, PRTL* formula $\phi_0$
output : $\mathcal{M}, \rho \models \phi_0$
function modelCheck($\mathcal{M}$, $\rho$, $\phi_0$):
    switch $\phi_0$ do
        case $true$: return true;
        case $p$: return $p \in L(last(\rho))$ ;
        case $\neg\phi$: return $\neg modelCheck(\mathcal{M}, \rho, \phi)$;
        case $\phi_1 \vee \phi_2$: return
          $modelCheck(\mathcal{M}, \rho, \phi_1)$ || $modelCheck(\mathcal{M}, \rho, \phi_2)$;
        case $\forall\psi$: return $findSatisfyingPrefixes(\mathcal{M}, \rho, \psi) = \{last(\rho)\}$;
        case $\mathsf{P}_{\psi}^{\bowtie q}$: return $prob(findSatisfyingPrefixes(\mathcal{M}, \rho, \psi)) \bowtie q$;
        case $\mathbb{G}_A\phi$: return $modelCheckAllGoals(\mathcal{M}, A, \rho, \phi)$;
        case $\mathbb{C}_A\phi$: return $modelCheckAllIntentions(\mathcal{M}, A, \rho, \phi)$;
        case $\mathbb{I}_A\phi$: return $modelCheckExistsIntention(\mathcal{M}, A, \rho, \phi)$;
        case $\mathbb{B}_A^{\bowtie q}\psi$: return
          $beliefWeightedExpectation(\mathcal{M}, A, \rho, V_{\mathbb{B},\mathcal{M},\psi}) \bowtie q$;
        case $\mathbb{CT}_{A,B}^{\bowtie q}\psi$: return
          $beliefWeightedExpectation(\mathcal{M}, A, \rho, V_{\mathbb{CT},\mathcal{M},B,\psi}) \bowtie q$;
        case $\mathbb{DT}_{A,B}^{\bowtie q}\psi$: return
          $beliefWeightedExpectation(\mathcal{M}, A, \rho, V_{\mathbb{DT},\mathcal{M},B,\psi}) \bowtie q$;
    end
end
```

to prefixes. This is what the function *prob* does. The way that novel cognitive and trust operators are handled is described below.

Cognitive Operators Model checking formulae of the form $\mathbb{G}_A\phi$, $\mathbb{I}_A\phi$, $\mathbb{C}_A\phi$ is mostly straightforward and follows a similar pattern for all three variants. For a given path, the algorithm iterates over possible/legal goal/intention changes and, for each, it model checks ϕ on a path obtained by appending that cognitive transition to the existing path. Finally, it aggregates those model checking results into a single outcome, depending on the type of quantification used in the operator semantics.

Case 1: $\phi_0 = \mathbb{G}_A\phi$. First, the algorithm retrieves all the possible goal changes of agent A using its synthesized goal strategy $\pi_A^g(\rho)$. Then, for each of them, it appends a cognitive transition representing that goal change to the current path ρ, obtaining ρs for some state $s \in \mathrm{S}$. With the extended path ρs, the algorithm model checks ϕ, i.e., tests whether $\mathcal{M}, \rho s \models \phi$. That step can be seen as a recursive call of the model checking algorithm. If any of those checks results in a negative answer, the algorithm terminates with a *false* too. If all of them succeed, the procedure outputs *true*. See Algorithm 6 for the formalised procedure.

Algorithm 6: Model checking a PRTL* formula of the form $\mathbb{G}_A\phi$

```
input  : ASMAS $\mathcal{M}$, agent $A$, a path $\rho$ in $\mathcal{M}$ and a PRTL* formula $\phi$
output : $\mathcal{M}, \rho \models \mathbb{G}_A\phi$
function modelCheckAllGoals($\mathcal{M}$, $A$, $\rho$, $\phi$):
    $goals \leftarrow synthesizeGoals(A, \rho)$;
    foreach goal $\in$ goals do
        $\rho' \leftarrow A.g(\rho, goal)$;      // $\rho'$ is $\rho$ extended by $A$'s goal change
        if $\neg modelCheck(\mathcal{M}, \rho', \phi)$ then
            return false;
        end
    end
    return true;
end
```

Case 2: $\phi_0 = \mathbb{I}_A\phi$. This case is completely analogous to the above, with goals replaced by intentions. Algorithm 7 formalises the steps required.

Algorithm 7: Model checking a PRTL* formula of the form $\mathbb{I}_A\phi$

```
input  : ASMAS $\mathcal{M}$, agent $A$ and a path $\rho$ in $\mathcal{M}$, PRTL* formula $\phi$
output : $\mathcal{M}, \rho \models \mathbb{I}_A\phi$
function modelCheckAllIntentions($\mathcal{M}$, $A$, $\rho$, $\phi$):
    $intentions \leftarrow synthesizeIntentions(A, \rho)$;
    foreach intention $\in$ intentions do
        $\rho' \leftarrow A.i(\rho, intention)$;  // $\rho'$ is $\rho$ extended by $A$'s intention
          change
        if $\neg modelCheck(\mathcal{M}, \rho', \phi)$ then
            return false;
        end
    end
    return true;
end
```

Case 3: $\phi_0 = \mathbb{C}_A\phi$. Model checking formulae of this form proceeds similarly to the two other cognitive operators, with few simplifications. First of all, rather than dealing with cognitive strategies, the algorithm operates on the legal intention function, which is predefined in the system. Moreover, the semantics of the $\mathbb{C}_A$ operator include existential quantification, and therefore the algorithm operates in a slightly different mode. In particular, for each legal intention change (as given by Int_A), the algorithm model checks ϕ against the extended path obtained by appending a cognitive transition to ρ. If that check is successful, the algorithm returns *true*. Otherwise, it continues to the next legal intention change. If none of the checks succeeds, the procedure outputs *false*.

Algorithm 8: Model checking a PRTL* formula of the form $\mathbb{C}_A\phi$

```
input  : ASMAS $\mathcal{M}$, agent $A$, a path $\rho$ and a PRTL$^*$ formula $\phi$
output : $\mathcal{M}, \rho \models \mathbb{C}_A\phi$
function modelCheckExistsIntentions($\mathcal{M}$, $A$, $\rho$, $\phi$):
    foreach intention $\in \omega_A^i(last(\rho))$ do
        $\rho' \leftarrow A.i(\rho, intention)$;  // $\rho'$ is $\rho$ extended by $A$'s intention
          change
        if $modelCheck(\mathcal{M}, \rho', \phi)$ then
            return true;
        end
    end
    return false;
end
```

Trust Operators We now describe procedures for model checking belief and trust operators. Note that all three operators are defined as belief-weighted expectation of some V-function defined in terms of the probability of satisfying a formula ψ. Therefore, in all cases, the algorithm starts by determining all paths ρ' that have the same observation as ρ, i.e., such that $\text{obs}_A(\rho') = \text{obs}_A(\rho)$. This can be done efficiently as long as an appropriate data structure is used to store observation functions of agents. An example would be made up of two dictionaries, one for efficient retrieval of an observation corresponding to a given state, and the other for looking up all the states mapped to a given observation. Then the algorithm computes the belief-weighted expectation of an appropriate V-function, by initialising a running sum to 0, followed by iterating over paths in the computed set, computing the value of the V-function for each path, multiplying it by its weight (which is given by the belief function) and adding it to the running sum (see Algorithm 9). After the loop is finished, the algorithm compares the result to q specified in the operator and returns *true* or *false*. Below, we describe how to compute the V-functions for each operator.

Case 1: $\phi_0 = \mathbb{B}_A^{\bowtie q}\psi$. This is the simplest case, since the V-function is simply the probability of the satisfaction of ψ, which is computed using *findSatisfyingPrefixes* (Algorithm 10).

Algorithm 9: Computing belief-weighted expectation of function V

```
input  : ASMAS $\mathcal{M}$, agent $A$, path $\rho$ and a function $V : \mathrm{FPath}^{\mathcal{M}} \to [0,1]$
output : $E_{\mathsf{be}_A}[V]$
1  function beliefWeightedExpectation($\mathcal{M}$, $A$, $\rho_0$, $V$):
2      $totalProbability \leftarrow 0$;
3      foreach $\rho \in \{\rho \mid \mathrm{obs}_A(\rho) = \mathrm{obs}_A(\rho_0)\}$ do
4          $totalProbability \leftarrow totalProbability + prob(\rho)$;
5      end
6      $result \leftarrow 0$;
7      foreach $\rho \in \{\rho \mid \mathrm{obs}_A(\rho) = \mathrm{obs}_A(\rho_0)\}$ do
8          $result \leftarrow result + V(\rho) \cdot \frac{prob(\rho)}{totalProbability}$;
9      end
10     return $result$;
11 end
```

Algorithm 10: Computing the value of function $V_{\mathbb{B},\mathcal{M},\psi}$

```
1  function $V_{\mathbb{B},\mathcal{M},\psi}(\rho)$:
2      return $prob(findSatisfyingPrefixes(\mathcal{M}, \rho, \psi))$;
3  end
```

Case 2: $\phi_0 = \mathbb{CT}^{\bowtie q}_{A,B}\psi$. This case is only a little more complicated, since it involves iterating over legal intention changes of B and computing probability of ψ for each, followed by taking the maximum or minimum, depending on what $\bowtie$ is. Again, the complexity of that task reduces to the computation of the probability of ψ (Algorithm 11).

Algorithm 11: Computing the value of function $V^{\bowtie}_{\mathbb{CT},\mathcal{M},B,\psi}$

```
1  function $V^{\bowtie}_{\mathbb{CT},\mathcal{M},B,\psi}(\rho)$:
2      $result \leftarrow (\bowtie = > \ || \ \bowtie = \geq)\ ?\ 0\ :\ 1$;
3      foreach $x \in \omega^i_B(last(\rho))$ do
4          $\rho' \leftarrow B.i(\rho, x)$;  // $\rho'$ is $\rho$ extended by $B$'s intention change
5          $p \leftarrow prob(findSatisfyingPrefixes(\mathcal{M}, \rho', \psi))$;
6          if $p \bowtie$ result then
7              $result \leftarrow p$;
8          end
9      end
10     return $result$;
11 end
```

Case 3: $\phi_0 = \mathbb{DT}^{\bowtie q}_{A,B}\psi$. This case in analogous to the above, with the exception that possible intention changes are used rather than legal ones, which does not affect the algorithm significantly (Algorithm 12).

Algorithm 12: Computing the value of function $V^{\bowtie}_{\mathbb{DT},\mathcal{M},B,\psi}$

```
function $V^{\bowtie}_{\mathbb{DT},\mathcal{M},B,\psi}(\rho)$:
    $result \leftarrow (\bowtie = > \ || \ \bowtie = \geq)\ ?\ 1\ :\ 0$;
    $intentions \leftarrow synthesizeIntentions(B, \rho)$;
    foreach $intn \in$ intentions do
        $\rho' \leftarrow B.i(\rho, intn)$;        // $\rho'$ is $\rho$ extended by $B$'s intention change
        $p \leftarrow prob(findSatisfyingPrefixes(\mathcal{M}, \rho', \psi))$;
        if $p \bowtie$ result then
            $result \leftarrow p$;
        end
    end
    return $result$;
end
```

4.5.3 The Algorithms in Action

We now put theory into practice by illustrating the execution of the model checking algorithms just presented. The formulas we use are set in the trust game example we introduced in Section 4.4. However, note that we do not strictly follow the state diagram from Figure 4.4; evaluation of certain formulas may require a different interleaving of temporal and cognitive transitions.

1. $\mathcal{T}, s_0 \models \mathbb{G}_{Alice}\mathbb{G}_{Bob}\mathtt{P}^{\leq 0.9}\Diamond^{\leq 1}(a_{Alice} = invest)$ This first formula expresses that regardless of Alice's and Bob's goals, the probability of her investing in the first round is no greater than 90%. First step of model checking this formula involves computing the support of Alice's goal strategy. However, recall that we have not given Alice's goal strategy in Section 4.4 due to our assumption that goals are static. The initial goal change acts as one-off setting of the goals, which are assumed to remain the same for the remainder of the game. We assume here that both players' initial goal strategy is uniform, so all legal goal changes are possible. Therefore, the model checking algorithm proceeds by verifying the subformula $\mathbb{G}_{Bob}\mathtt{P}^{\leq 0.9}\Diamond^{\leq 1}(a_{Alice} = invest)$ when execution history is s_0s_1 and s_0s_2, and, subsequently, the subformula $\mathtt{P}^{\leq 0.9}\Diamond^{\leq 1}(a_{Alice} = invest)$ for all execution histories of length two. Since Alice invests with probability 30% when she is passive and with probability 90% when she is active, the formula is true for every execution history and hence the original formula is verified to hold.

2. $\mathcal{T}, s_0 \models \neg\mathbb{G}_{Alice}\neg\mathbb{G}_{Bob}\mathbb{C}_{Bob}\forall\Box^{\leq 2}\neg richer_{Bob,Alice}$ The second formula we model check expresses that Alice has a goal such that whatever mental state Bob chooses, he will not become richer than Alice within two steps. The atomic proposition $richer_{Bob,Alice}$ is true in states where Bob has earned more money than Alice. The algorithm starts in the same way as for the previous formula - by iterating over Alice's possible goal changes in the initial state (but this time the result will have to be negated reflecting the presence of $\neg$ operator). We again assume that both goals are in $\text{supp}(()\pi^{g}_{Alice}(s_0))$. For each possible goal change, the algorithm continues by considering possible goals of Bob (we assume both are possible) and his legal intentions. The PCTL* formula $\forall\Box^{\leq 2}\neg richer_{Bob,Alice}$ is evaluated at every path obtained by combining all those cognitive transitions.

To process of evaluating that subformula boils down to finding prefixes that satisfy $\Box^{\leq 2}\neg richer_{Bob,Alice}$. Now, Bob is richer than Alice as soon as she invests her money with him, so any path that has Alice investing will not satisfy the formula. However, regardless of her goal, Alice's action strategy assigns positive probability to investing, meaning a state where $richer_{Bob,Alice}$ holds can be reached within two steps. Hence $\forall\Box^{\leq 2}\neg richer_{Bob,Alice}$ is false at any execution history that has set mental states of agents, which makes the original formula false.

2'. $\mathcal{T}, s_0 \models \neg\mathbb{G}_{Alice}\neg\mathbb{G}_{Bob}\mathbb{C}_{Bob}\mathsf{P}^{\geq 0.7}\Box^{\leq 2}\neg richer_{Bob,Alice}$ Next, we briefly comment on what changes if we replace the universal operator $\forall$ with a probability operator $\mathsf{P}^{\geq 0.7}$. The model checking algorithm starts as before and the prefixes satisfying $\Box^{\leq 2}\neg richer_{Bob,Alice}$ are the same. However, this time, the probability of those prefixes matters. Since paths satisfying the formula are ones where Alice withholds the money, the probability of those prefixes is equal to the probability of Alice selecting that action. This is equal to 0.7 when she is passive and 0.1 when she is active, regardless of Bob's mental state. Hence the formula is true.

3. $\mathcal{T}, \rho \models \mathbb{CT}^{>0.5}_{Alice,Bob}\bigcirc(invest_{Alice} \to \bigcirc share_{Bob})$ We now turn our attention to trust operators; we begin by considering Alice's trust in Bob's competence to share the money upon investment. We assume atomic proposition $invest_{Alice}$ is true in states in which $a_{Alice} = invest$ (and similarly for $share_{Bob}$). We also assume that the formula is evaluated at a path ρ at which goals of agents are set. In particular, we use the state diagram from Figure 4.4 and set $\rho = s_0 s_2 s_5$, i.e. Alice is active while Bob is an investor – but Alice does not know that. Instead, she evaluates the belief-weighted expectation of an appropriate V-function, where her belief is given by her goal preference function. We assume it is the same as defined in Section 4.4, i.e., uniform, yielding the following belief:

$$\mathtt{be}_{\text{Alice}}(\rho) = 1/2,$$
$$\mathtt{be}_{\text{Alice}}(\rho') = 1/2,$$

where $\rho' = s_0 s_2 s_6$ is the path Alice cannot distinguish from ρ.

Let $\psi = \bigcirc(invest_{Alice} \to \bigcirc share_{Bob})$. Then the value of $V^{>0.5}_{\mathbb{CT},\mathbb{T},Bob,\psi}(\rho)$ is computed according to Algorithm 11, by considering each legal intention of Bob in turn and computing the probability of ψ being satisfied. Since the relational symbol is $>$, operator computes the upper bound, so the maximum probability is recorded. The formula holds when Alice's investment is followed by Bob sharing the money; recalling Bob's strategy given in Table 4.3, the probability of this is 1 when Bob's intention is to share and 0 otherwise. Hence $V^{>0.5}_{\mathbb{CT},\mathbb{T},Bob,\psi}(\rho) = 1$. The value of the V-function is the same at path ρ' since both intentions are legal regardless of Bob's goal. Therefore, the formula is verified to hold at path ρ.

4. $\mathcal{T}, \rho \models \mathbb{DT}^{>0.5}_{Alice,Bob}\bigcirc(invest_{Alice} \to \bigcirc share_{Bob})$ This time, we substitute competence trust operator with disposition trust, which describes Alice's belief that Bob is willing to share the invested money. We set $\rho = s_0 s_2 s_5$ as before. The aspect that is different now is the V-function, which is now computed with respect to possible, rather than legal, intentions. This necessitates an invocation of Bob's pro-attitude synthesis, as indicated in Algorithm 12. Recalling the guarding mechanism given for Bob in Equations 4.2, synthesising Bob's intention boils down to computing his belief over Alice's goal. Based on Bob's goal preference function (Equation 4.5), we compute his belief as

$$\mathtt{be}_{\text{Bob}}(o_{\text{Bob}}, \rho) = 2/3,$$
$$\mathtt{be}_{\text{Bob}}(o_{\text{Bob}}, \rho_1) = 1/3,$$

where $\rho_1 = s_0 s_1 s_3$ is another path Bob cannot distinguish from ρ and o_{Bob} is the common observation associated to both paths. Similarly,

$$\mathtt{be}_{\text{Bob}}(o'_{\text{Bob}}, \rho') = 2/3,$$
$$\mathtt{be}_{\text{Bob}}(o'_{\text{Bob}}, \rho'_1) = 1/3,$$

where $\rho' = s_0 s_2 s_6$ as before, $\rho'_1 = s_0 s_1 s_4$ is another path Bob cannot distinguish from ρ' and o'_{Bob} is the common observation.

In any case, $\mathbb{B}^{\geq 0.7}_{Bob}\, active_{Alice}$ does not hold since Bob's conviction that Alice is active is not strong enough. Hence, Bob's intention is synthesised as *keep* regardless of his goals (when Bob is an opportunist, synthesis is trivial). Under that intention, probability of satisfying paths is computed according to Algorithm 4 as 0. Formally,

$$V^{>0.5}_{\mathbb{DT},\mathcal{T},Bob,\psi}(\rho) = 0,$$
$$V^{>0.5}_{\mathbb{DT},\mathcal{T},Bob,\psi}(\rho') = 0.$$

In fact, the result would be the same even if the $>$ operator were replaced by $<$, since only one intention is possible. Therefore, the formula does not hold at ρ; in fact, Alice has no trust in Bob sharing his profits in case she invests.

4.6 Critique

We conclude this chapter by taking a critical look at the ASMAS framework. This will partly serve as motivation for an alternative trust model we present in this thesis (Chapter 5). We list the deficiencies of ASMASs below and give a brief overview of each of them.

High Complexity The tuple defining ASMASs has eleven components, three of which are novel. This presents a significant challenge to familiarising oneself with the workings of the framework, as the reader of this thesis might have found out the hard way. However, the problems do not end there. Once acquainted with its mechanisms and keen to use the model in practice, one must specify all the components first, as we do in Section 4.4. This involves assigning goals and intentions to agents, a non-trivial task, which is not easily automated. Second, for each intention, a corresponding action strategy must be given; no guidance is given on how such a strategy should be determined. Additionally, preference functions of agents must be provided; these could conceivably come from data, although the actual mechanism is not immediately clear. Finally, pro-attitude synthesis, a very elegant concept and one of the key elements of the framework, is formulated in terms of custom rules (the guarding mechanism) which must be given as input. Both action strategies and guarding functions involve numerical values (probabilities of different actions and belief/trust thresholds) which are not easy to reliably determine, but even small variations may lead to significant differences in evaluations of logical operators, as we have seen towards the end of Section 4.4.

The modelling of the cognitive dimension is subject to several conventions and assumptions that are not strictly enforced; this includes the interpretation of what goals are and their static nature, the fact that intention changes precede taking actions and the way goals influence intentions. Moreover, the distinction of legal and possible goals and intentions may be appropriate for certain robotic settings, but, in most scenarios, it only serves to complicate an already complex model.

Deficient Trust Definitions Next, we analyse a key aspect of the framework, namely, the definitions of trust operators. The mathematical formulation is very complex and, arguably, hardly intuitive. Two different notions of trust are defined, and the framework only supports the computation of lower and upper bounds on trust values. As our trust game example shows, those bounds may often be quite wide and not particularly useful. In any case, an upper bound can be interpreted as the probability of an event (represented by a path formula) when trustee aims to maximise it, while a lower bound expresses this probability when being minimised by trustee. However, what is lacking is a quantitative indication of which of the two paths the trustee is going to take. Also, the definitions may be described as lacking robustness, in the sense that a small change in an agent's intention strategy may cause trust bounds to change significantly, as demonstrated as part of the trust game example. We also point out a significant conceptual limitation, namely that A's disposition trust towards B is computed with respect to B's intention strategy. In reality, A will rarely possess such information, in particular not in a competitive scenario such as the trust game. In the framework, this is rectified to a degree by taking the support of the distribution, thereby providing only partial information to the trustor. However, this results in the issues mentioned above, such as the lack of robustness and uninformative bounds.

Human Incompatibility Several aspects of the framework make it inappropriate for reasoning about the behaviour of humans. Firstly, as discussed above, the semantics of trust operators rests on fairly dubious assumptions. One's trust towards another should be based purely on information available to them, which the other agent's plan of action is generally not. Second, the model is discrete in nature, with a finite set of goals and intentions (and hence action strategies). However, humans display a continuum of behaviour and personalities that cannot be captured accurately with a discrete framework. Moreover, as Daniel Kahneman argues in his book [107], homo sapiens rarely carry out fully strategic reasoning; rather, they act on the spot, based on limited information they have. Using nomenclature introduced by the author, “System 1”, representing the fast, emotional, intuitive thinking usually prevails over the effortful, logical deliberations characteristic of “System 2”. Humans use mental shortcuts to make sense of complex issues and often act irrationally. This process cannot be reduced to an action strategy.

Lack of Scalability Figure 4.4 shows how the addition of a cognitive mechanism to a simple game increases its state space severalfold. The more soundly one wishes to model a given system, the more goals, intentions and action strategies one must define, further increasing the scale of state space expansion. Similarly, the more actions there are available to agents, the more complex the action strategies become and more intentions are required to model a range of behaviours. In fact, the trust game serves as a good example of this shortcoming. In Section 4.4, we have presented its basic version, characterised by binary decisions of agents. However, in the literature, it is much more common to assume that agents can send integer amounts of dollars not exceeding their endowment and that the game lasts for several rounds. Attempting to model that variant with an ASMAS would require a substantial number of intentions to be defined to represent the various strategies agents may adopt.

4.7 Conclusion

In this chapter, we have presented autonomous stochastic multi-agent system, a quantitative framework for reasoning about social trust. We have considered the problem of model checking formulae built using novel logical operators and proposed algorithms that allow one to verify (bounded) formulae automatically. We have also identified weaknesses of ASMAS and argued that the definition of trust proposed therein does not capture the human notion of trust well. We address those weaknesses in Chapter 5.

5 Cognitive Stochastic Multiplayer Games – Syntax & Semantics

Contents

This chapter presents the cognitive stochastic multiplayer game (CSMG) model, which embodies our attempt to provide a framework that accurately captures human mental processes, permitting a well grounded formulation of trust. Unlike ASMAS, which is based on a verification setting, appropriate for checking formal properties of systems, CSMG draws from game theory, which is more applicable to modelling interactions of self-interested agents.

5.1 Motivation

The design choices behind CSMG are largely motivated by the deficiencies of ASMAS. Recall that, in ASMAS, even though partial observability allows one to model differences in knowledge of different agents, trust definitions are based on implicit assumptions of one agent knowing another agent's strategy. Therefore, in CSMG, we are very careful to accurately model what an agent knows and what they do not know. In the latter case, cognitively-inspired mechanisms are given that model how agents estimate what they cannot observe.

Next, CSMG addresses the discreteness of ASMAS. We argue that human personalities fall on a spectrum and cannot be captured reliably with a finite set of goals and intentions. In the context of the trust game example, most people cannot be categorised as an opportunist or an investor; rather, they will be opportunistic to a certain extent. Therefore, CSMG postulates that each (human) agent is driven by a variety of goals that compete with each other to steer agent's actions one way or another. What determines agent's personality are the relative weights that they assign to various goals; crucially, these are points in a multidimensional, continuous space.

Further, having learned the lesson of ASMAS's complexity spinning out of control, our aim when designing CSMG was to keep things simple. Of course, human mental processes are highly complex and one cannot avoid intricacy altogether. However, a user-friendly framework should minimise the number of parameters and inputs required to use it. We also strive to simplify the overall mechanism of the model by separating an easy to comprehend base and more advanced, but non-essential, components.

Despite our model being grounded in game theory, we deviate from the mainstream approach of computing Nash equilibria to predict game outcomes. We believe NE theory is better suited to highly strategic, often politically-inspired interactions, characterised by high consequence and long deliberation. Examples include Brexit negotiations, Cuban Missile Crisis or Russia's invasion of Ukraine. In contrast, we argue that everyday, casual decisions of humans are driven by simpler, often approximate, reasoning. Given multiple sources of uncertainty, computing a Nash equilibrium might be challenging for a machine, let alone a *homo sapiens*. Moreover, Goeree et al. [108] show how simple modifications of the payoff structure lead to large inconsistencies between theoretical, NE-based predictions and observed human behaviour.

Moreover, we deemed it necessary that our framework be implementable, to allow practical applications such as its use in robots. Due to the probabilistic nature of human belief and reasoning process, probabilistic programming is highly appropriate for realising theoretical formulations of our framework, as we show in Section 6.1.

Related to its implementability, in accordance with recent trends, our goal was to support inference of agent parameters from data, a mechanism missing from ASMAS. This is greatly facilitated by choosing a probabilistic programming implementation, which comes with a powerful inference mechanism.

Further motivation for our framework is provided by a tournament for prediction of human choices organised by Plonsky et al. [109]. The authors conclude that a combination of descriptive, behavioural models and machine learning algorithms is the most promising recipe for successfully predicting human behaviour. Further, they suggest that quantitative models that integrate multiple behaviour-affecting mechanisms are superior to frameworks that focus on specific aspects of decision making.

5.2 Overview

Having described the motivation for proposing our framework and our main design choices, we now overview its key components and narrow down the setting our model applies to.

Throughout any given day, week, month or year, humans make many decisions, ranging from the trivial ones such as what to eat for breakfast or what socks to wear, to the more serious ones, such as which car to buy or which university course to apply to. Depending on the importance and consequence of a decision, a different mechanism may be employed to select a course of action.

In this work, we restrict our attention to choices the are made on the spot, without extended deliberation. We focus on atomic decisions, i.e., ones that involve a single choice between well-defined alternatives, rather than necessitating a multi-stage plan.

Our framework builds on game theory, which has traditionally been used to analyse interactions of rational agents. However, behavioural predictions offered by standard game-theoretic models rest upon the notion of equilibrium. When people find themselves in a given scenario for the first time, and have incomplete knowledge about their opponents, arriving at an equilibrium is doubtful.

Instead, we postulate that agents make decisions by considering the available alternatives, along with possible future developments, and evaluating each according to their preferences, which are captured by a utility function. Now, this may be considered a contentious assumption, given the body of work that supposedly falsifies expected utility theory [98]. However, one of our main novel contributions is the way that the utility of an agent is formulated; in particular, it captures not only physical quantities such as money or other possessions, but also agents' emotions, which strongly affect behaviour.

Besides that, reasoning limitations of homo sapiens, such as bounded rationality and finite future outlook, are represented in our formalism. Hence, our framework integrates existing behavioural models, such as quantal response [96] and Level-k models [110], consistently with recommendations of Plonsky et al. [109]. Further, our model supports encoding of heuristics people use to simplify and make sense of reality [111, 112], particularly to estimate what they do not know about others.

5.3 Model Description

In this section, we formally give the definition of our model and describe in detail all its components. We illustrate our construction with a following running example.

Example. Tic-tac-toe is a two-player pen and paper game played on a 3x3 grid in which players take turns to insert an 'X' (the cross player) or an 'O' (the nought player) into an empty square. A goal of each player is to fill a full row, column or a diagonal with their mark ('X' or 'O'). Figure 5.1 shows a possible play of tic-tac-toe; note that the cross player always starts.

There exist many strategies for either player which guarantee at least a draw, and they are not hard to follow. Hence, a game of tic-tac-toe between two players following optimal strategies will always end in a draw. In reality, however, this is not always the case. Children, especially the younger ones, often find tic-tac-toe challenging, as their cognitive limitations prevent them from playing perfectly.

In a matchup between a parent and their kid, the adult should be a clear favourite. However, as some readers may know from experience, the outcome of such a game is far from decided; in fact, the child wins more often than not. In what follows, we use our framework to explain this apparent paradox.

A cognitive stochastic multiplayer game (CSMG) is a tuple $(\mathrm{Ags}, \mathrm{S}, \{\mathrm{Act}\}_{A \in \mathrm{Ags}}, \mathrm{T}, \mathrm{R}, \mathrm{P}, \{\vec{\lambda}^A\}_{A \in \mathrm{Ags}}, \{\mathrm{be}^0_\mathrm{A}\}_{A \in \mathrm{Ags}}, \{\theta_A\}_{A \in \mathrm{Ags}}, \{\mathrm{est}^0_\mathrm{A}\}_{A \in \mathrm{Ags}})$, where $(\mathrm{Ags}, \mathrm{S}, \{\mathrm{Act}\}_{A \in \mathrm{Ags}}, \mathrm{T}, \mathrm{R})$ is a turn-based SMG as introduced in Section 3.4.3. Standard concepts introduced for SMGs, such as a set of finite paths FPath, functions ACTIONS and OWNS, and path notations remain the same in CSMGs. Novel components represent mental reasoning of agents; in particular, P is a mental counterpart of the physical rewards structure R, $\{\vec{\lambda}^A\}_{A \in \mathrm{Ags}}$ is a set of goal coefficients of each agent, $\{\mathrm{be}^0_\mathrm{A}\}_{A \in \mathrm{Ags}}$ records initial beliefs of agents, $\{\theta_A\}_{A \in \mathrm{Ags}}$ are agents' meta-parameters and $\{\mathrm{est}^0_\mathrm{A}\}_{A \in \mathrm{Ags}}$ represents agents' initial estimations of their opponents' mental states.

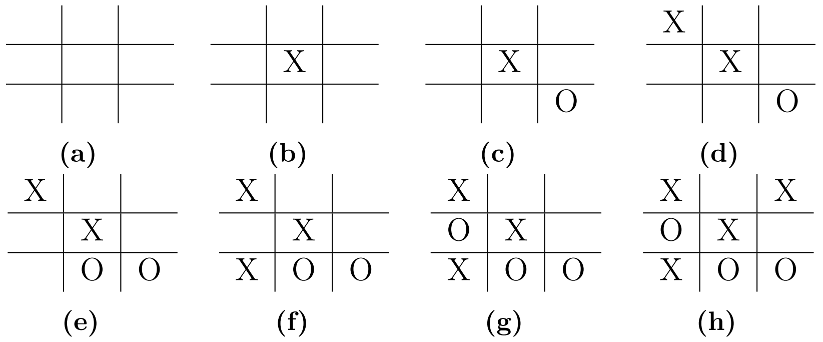


Figure 5.1: An example game of tic-tac-toe, developing (a) to (h)

Example. Tic-tac-toe between a parent and their child is modelled by setting $\mathrm{Ags} = \{\mathrm{kid}, \mathrm{parent}\}$ (abbreviated to k and p in subscripts); the set of actions $\mathrm{Act}_\mathrm{A} = \{(x, y) \mid x, y \in \{0, 1, 2\}\}$ of each player consists of nine elements, each identifying a single position in the grid (e.g., Figure 5.1f adds an 'X' at position $(2, 0)$); each state records the history of execution (e.g., state from Figure 5.1d is $[(1, 1), (2, 2), (0, 0)]$); the transition function encodes the informal description of the game (we assume kid is the cross player and goes first); there is one physical reward structure $\mathrm{R}_1 = \{\mathrm{r}_{1,\mathrm{k}}, \mathrm{r}_{1,\mathrm{p}}\}$ that assigns no action rewards, but allocates non-zero rewards in states in which the game has ended: $+1$ for the winner and -1 for the loser. Note that tic-tac-toe is a turn-based game.

In what follows, we introduce in detail the novel elements of the CSMG definition and motivate them by referring to the running example.

Example. A standard approach to modelling players' motivations in a game like tic-tac-toe is to assign a positive reward for winning the game and a negative reward for losing. However, it is clear that when parent plays against their child, something else is at stake. One may be tempted to modify the reward structure by changing the parent's rewards to equal child's rewards, so that the parent prefers losing to winning. However, this would be too simplistic and might lead to unrealistic behaviour, whereby the parent never defends against obvious threats and plays unnatural moves.

Instead, we postulate that, rather than trying to increase the probability of child winning, the parent wants to increase child's satisfaction from playing the game. The difficulty that arises is that, unlike the outcome of the game, satisfaction constitutes private information of each player. Therefore, it cannot be captured by the standard reward structure R_1, which deterministically assigns rewards at each state. Instead, we treat satisfaction, and other mental states, differently.

5.3.1 Mental Rewards

In general, our observation is that humans are motivated not only by physical, easily quantifiable quantities, such as money or time, but they are driven also by mental goals. Examples include high-level desires such as developing relationships with people, making others happy or causing an enemy to suffer, as well as emotional considerations: avoiding guilt or shame, maximising trust and pride. Distinctive features of these mental states are that they are difficult to measure and quantify and typically not observable to others – which complicates things further, given that an agent might be motivated by someone else's mental state (e.g., I may care how much you trust me).

To represent mental goals, we equip each agent with a set of *mental variables* that are latent in a sense that other agents do not know their values. Each mental variable represents a mental state or an emotion, such as anger, joy, regret or pride. An agent knows how they feel, and so they can evaluate their mental state at any point – such evaluation might take agent's belief and personality into account. On the other hand, mental states of others are unknown to an agent, which necessitates estimation of their values. The intuition behind mental rewards is to assign a numeric value to those feelings experienced by the agents, so that they can be included in agent's utility function.

Before diving into detail, a note about nomenclature: *mental state* is the emotion we are considering (trust, guilt, shame etc.); *mental goal* is closely related, but captures what an agent is trying to achieve with respect to some mental state (e.g., minimize guilt or increase trust); *mental variable* is purely syntactic – it identifies a mental state of an agent; *mental reward* is a numerical value assigned to a mental state, appropriate for inclusion in a utility function.

Formally, along with a set of reward structures R, which model physical rewards, CSMG includes a set of mental reward structures $\mathrm{P} = \bigcup_{i=1}^{l}\{\mathrm{P_i}\}$, one for each mental state we wish to model (we assume there are l such). Each mental reward structure $\mathrm{P_i}$ is a tuple $(\{\eta_i^A\}_{A\in\mathrm{Ags}}, \delta_i, \omega_i)$, where η_i^A is the i^{th} mental variable of an agent $A \in \mathrm{Ags}$, while δ_i and ω_i are used to compute mental rewards.

In particular, $\delta_i : \mathrm{FPath} \times \mathrm{Act} \times \mathrm{Ags} \times \mathrm{Ags} \to ([-1,1] \to [-1,1])$ is a mental state dynamics model, which encodes agents' appraisal of mental states of others. Given a path $\rho \in \mathrm{FPath}$, $a \in \mathrm{Act}$ and $A, B \in \mathrm{Ags}$, $\delta_i(\rho, a, A, B)$ is a dynamics function that encodes a heuristic according to which A estimates how i^{th} mental state of B changes when action a is taken in the last state of ρ. For instance, Bob may hypothesize that being late for his weekly meeting will deteriorate his relationship with his supervisor, unless that relationship is already at a low point, in which case it may get better as he at least showed up. In some cases, the dynamics model is constructed using common sense, while at other times research findings from fields of psychology and cognitive sciences are utilised. It is also possible to learn the dynamics function from data using machine learning techniques. Dynamics functions are typically continuous (but we do not enforce it) and may be partial, as some mental states only take nonnegative values (trust, for example).

On the other hand, mental state evaluation function, $\omega_i : \mathrm{FPath} \times \mathrm{Ags} \to [-1, 1]$, captures the mechanism through which an agent experiences their own mental state. For example, it may be that the supervisor has already given up on Bob due to his past behaviour and the relationship cannot be recovered in her eyes. This may be because the supervisor is unforgiving, which Bob might not be aware of. In general, evaluation function ω_i may take into account personality and beliefs of an agent and its definition will often be informed by insights from relevant disciplines.

Example. Coming back to our tic-tac-toe example, satisfaction is the only mental state we model, represented by mental variables η_1^k (child's satisfaction) and η_1^p (satisfaction of the parent). The mechanism through which the parent makes inferences about the value of η_1^k is the satisfaction dynamics function δ_1. It encodes a heuristic that dictates how satisfaction changes as the game develops and it is specified in terms of forks and blunders, which are certain types of moves in tic-tac-toe, easily explained with an example.

For a fork, see Figure 5.1f – the cross player now has two winning lines (left column and a diagonal), which guarantees victory in the next move (assuming optimal play). Hence, a fork is a move that creates two distinct winning opportunities for oneself, while not giving any winning opportunity to an opponent. When it comes to blunders, there are two types: (i) not defending against an obvious threat (any move other than $(3, 1)$ in the state from Figure 5.1e is a blunder of that type), (ii) not taking an obvious winning opportunity (e.g., any move other than $(1, 3)$ in the state from Figure 5.1g).

With that, the heuristic captured by satisfaction dynamics postulates that winning the game increases satisfaction, especially if the game was won as a consequence of creating a fork; however, satisfaction is lower if opponent blundered. On the other hand, losing the game brings about a decline in satisfaction and the reduction is much steeper if losing is a result of own blunder. For a precise specification, see Algorithm 13. Note that concrete numerical values are used to represent changes of satisfaction as a result of various actions; in a real application, these values would ideally come from data, but for the purposes of our example, we have chosen them fairly arbitrarily.

We omit a specification of satisfaction evaluation function, as it does not feature in agents' decision making (which is explained in what follows, once utility functions of agents are introduced).

Algorithm 13: Satisfaction dynamics

```
input  : state, action, estimator, estimatee, satisfaction
output : estimator's updated estimation of estimatee's satisfaction after
         action is taken in state
function updateSatEst(state, action, estimator, estimatee, satisfaction):
    /* only model parent's estimation of kid's satisfaction   */
    if $estimator \neq parent$ or $estimatee \neq child$ then
        return 0;
    end
    /* determine whose turn it is at state                    */
    $turn = \textsc{owns}(state)$;
    if $turn = kid$ then
        if action is a fork then
            return MIN(1, satisfaction + 0.7)
        end
        if action is a blunder then
            return MAX(1, satisfaction - 1)
        end
        if action wins the game then
            return MIN(1, satisfaction + 0.3)
        end
    else
        if action is a blunder then
            return MAX(-1, satisfaction - 0.5)
        end
        if action wins the game then
            return MAX(-1, satisfaction - 0.3)
        end
    end
    return satisfaction;
end
```

5.3.2 Agent Characteristics

So far, we have introduced one novel component of CSMGs, a set of mental reward structures P that capture how emotions of agents affect their behaviour. The intuition is that an agent generally receives various rewards, some of them physical and some of them mental, but they value some more than others.

To represent this, each agent in our framework is equipped with a vector of *goal coefficients*, which provides subjective weights of various reward structures. This vector, referred to also as *agent characteristics* and denoted $\vec{\lambda}^A$ (for agent A), gives rise to the $\{\vec{\lambda}^A\}_{A\in\mathrm{Ags}}$ component in our definition. Goal coefficients encode players' preference over various rewards. Each vector $\vec{\lambda}^A$ consists of $k + ln$ nonnegative real numbers, where $k \in \mathbb{N}$ is the number of physical rewards ($|\mathrm{R}|$), $n \in \mathbb{N}$ is the number of agents ($|\mathrm{Ags}|$) and $l \in \mathbb{N}$ is the number of mental states (see Section 5.3.1). Intuitively, each agent cares about physical rewards as well as mental state of every other agent, including themselves (Section 5.4.1 formalises this assumption). Without loss of generality, we assume that entries of $\vec{\lambda}^A$ sum to 1.

5.3.3 Belief

A vector of goal coefficients, which reflects personality of a player, constitutes private information of an agent. A discrepancy between actual characteristics of an agent and opponents' perception of these characteristics is captured by the notion of belief. To express it, we introduce a probability space on the set of possible goal coefficient vectors, $\Lambda = \{\vec{\lambda} \mid \sum_i \lambda_i = 1\}$, of agents and, for each $A \in \mathrm{Ags}$ and a path $\rho \in \mathrm{FPath}$, a belief function $\mathrm{be}^\rho_\mathrm{A} : \Lambda \to \mathbb{R}$, which acts as a density function on Λ. Then, a probability space of agent A, after executing ρ, is a triple $(\Lambda, \mathcal{F}_A, \mathrm{Pr}^\rho_\mathrm{A})$, where the set of events $\mathcal{F}_A$ is a standard Borel σ-algebra generated by open balls in Λ and the probability measure $\mathrm{Pr}^\rho_\mathrm{A} : \mathcal{F}_A \to [0, 1]$ is given in terms of belief by $\mathrm{Pr}^\rho_\mathrm{A}(X) = \int_X \mathrm{be}^\rho_\mathrm{A}(\vec{\lambda})d\vec{\lambda}$.

We assume agents update their belief in a Bayesian way every time their opponent takes an action. That requires initial, i.e., prior, belief of every agent to be specified, represented by the $\{\mathrm{be}^0_\mathrm{A}\}_{A\in\mathrm{Ags}}$ component of a tuple defining CSMGs. A formal specification of belief update is deferred until Section 5.4.3, but the intuition is that observing an action taken by an opponent provides an agent with an insight of what values of opponent's goal coefficients are more likely than others. This insight is incorporated into agent's current belief to produce an updated belief.

Example. In tic-tac-toe, the parent has two sources of motivation (each represented by a reward structure): winning the game and maximising their child's satisfaction. Therefore, the parent's characteristics is specified by a two element vector $\langle\lambda^p_1, \lambda^p_2\rangle$. We assume that they assign significantly more importance to the latter incentive, i.e., $\lambda^p_1 < \lambda^p_2$.

However, the child is not aware of what their parent is up to – if they were, the game would not be entertaining for them. Hence, the child will think that $\lambda^p_1 = 1$, which can be formally captured using a Dirac distribution.

5.3.4 Meta-Parameters

Apart from a vector of goal coefficients, each agent is characterised also by a set of meta-parameters, so called because they control agents' decision-making process on a higher level. One such is lookahead $\beta \in \mathbb{N}$, which determines how far into the future a player looks when computing their action. Other meta-parameters include rationality $\alpha \in [0, \infty]$, which comes into play as a parameter of the softmax choice formula introduced in Section 3.4.3, and discount factor $\gamma \in (0, 1]$, which defines how much an agent discounts future rewards. The set of all meta-parameters is denoted Θ. The exact way meta-parameters regulate agents' decision-making process is described in Section 5.4.2.

Example. Lookahead is especially important in tic-tac-toe and popular board games such as chess or Go. One of the distinctive characteristics of advanced players is their ability to simulate the execution of the game multiple moves into the future, allowing them to evaluate a given choice better. In tic-tac-toe, we would expect the child to have a lookahead no greater than three, while for a parent a value of six or seven would not be uncommon. Rationality would also differ significantly; we expect the kid to err in their computations more often than their parent, which we could model by setting $\alpha_k = 5$ and $\alpha_p = 20$. Finally, discounting is useful to reflect the fact that players typically prefer to win the game sooner rather than later – any value of γ smaller than one does the trick.

In our framework, nested reasoning of agents necessitates that they quantify the likelihood of different configurations of meta-parameters of their opponent, captured by the $\{\theta_A\}_{A \in \mathrm{Ags}}$ component in the definition. In particular, for each agent $A \in \mathrm{Ags}$ and their opponent $B \in \mathrm{Ags}$, we take $\Theta_B = [0, \infty] \times \mathbb{N} \times (0, 1]$ to be the set of possible meta-parameter tuples of B. Then, $\theta_A : \Theta_B \to \mathbb{R}$ satisfying

$$\int_{\Theta_B} \theta_A(x)dx = 1$$

(i.e., θ_A is a probability density function in Θ_B) describes A's estimation of B's meta-parameters.

5.3.5 Mental State Estimation

Recall that a mental variable of an agent, such as η_1^k from our model of tic-tac-toe, identifies that agent's mental state (in this case, satisfaction of the child). At any point of execution, the value of agent's A i$^{\text{th}}$ mental state is denoted $[\![\eta_i^A]\!]^\rho$ (for current path ρ) and computed using the mental state evaluation function $\omega_i(\rho, A)$.

However, an agent cannot directly compute the value of their opponent's mental state. Instead, they start with an estimation (which takes the form of a probability distribution) and they update it according to the dynamics model as the execution progresses.

To express it formally, we introduce some additional notation. For a function $f : X \to Y$ and an element $y \in Y$, $f^{-1}(y)$ denotes the preimage of y, i.e., the set $\{x \in X \mid f(x) = y\}$. Further, for an agent $B \in \mathrm{Ags}$, let $\mathrm{H_B} = \bigcup_{1 \le i \le l} \{\eta_i^B\}$ be the set of mental variables of B. Then, for $A, B \in \mathrm{Ags}$ and $\rho \in \mathrm{FPath}$, the *mental state estimation function* $\mathrm{est}_\mathrm{A}^\rho : \mathrm{H_B} \to \mathcal{D}^f([-1,1])$ gives A's estimation, expressed as a probability distribution, of each mental state $\eta_i^B \in \mathrm{H_B}$ of B, upon executing path ρ. Recall that $\mathcal{D}^f([-1,1])$ denotes the set of probability distributions on $[-1,1]$ with countable support.

Letting $\rho = s_0 a_0 \ldots s_n a_n s_{n+1}$ and fixing $\eta_i^B \in \mathrm{H_B}$, $\mathrm{est}_\mathrm{A}^\rho(\eta_i^B)$ is computed recursively by A, starting from their initial estimations $\mathrm{est}_\mathrm{A}^0 : \mathrm{H_B} \to \mathcal{D}^f([-1,1])$ and using the dynamics function δ_i, as follows:

$$\mathrm{est}_\mathrm{A}^\rho(\eta_i^B)(x) = \begin{cases} \mathrm{est}_\mathrm{A}^0(\eta_i^B)(x) & \text{if } n+1 = 0, \\ \sum_{y \in \delta_i(\rho[0 \ldots n], a_n, A, B)^{-1}(x)} \mathrm{est}_\mathrm{A}^{\rho[0\ldots\mathrm{n}]}(\eta_i^B)(y) & \text{otherwise.} \end{cases}$$

Note that restricting the codomain of the mental state estimation function to distributions with countable support ensures that a summation may be used in the above identity. Otherwise, various restrictions would have to be placed on the dynamics functions δ_i to ensure that sections of the estimation function can be integrated. This would add unnecessary complexity to our framework – as explained in Chapter 6, discrete probability distributions are preferred over continuous ones anyway.

With that, agent's A expectation of the value of i^{th} mental state of B is computed with respect to A's estimation function as

$$\mathbb{E}_A^\rho[\eta_i^B] = \sum_{x \in \mathrm{supp}(\mathrm{est}_\mathrm{A}^\rho(\eta_i^B))} x \cdot \mathrm{est}_\mathrm{A}^\rho(\eta_i^B)(x).$$

5.4 Model Semantics

Having introduced the components of our model, we now describe how they combine to give its semantics, which boils down to a formal specification of agents' decision-making process. The first step in that direction is the definition of a *cognitive utility function.* Then, a method of computing agent's own, and their opponents', expected utility is given. However, due to multiple sources of partial observability, which differ between agents, special care has to be taken to truthfully capture their cognitive processes.

Note that our formulation of the decision-making process may be used in two ways. Typically, we use our model "forwards" to generate behavioural predictions, in the form of posterior predictive distribution over possible decisions. However, it is also possible to use the framework "backwards", to infer characteristics of agents given behavioural data. We describe both of those modes of operation in more detail in Section 6.1.

5.4.1 Utility Function

Cognitive utility function extends the standard one, defined in Section 3.4.3, with mental rewards. Note that, due to the nature of mental rewards, we record their values in states, but do not associate any mental rewards with actions. Therefore, action utility remains the same in the cognitive model. Moreover, we allow for nonlinearity of utility by introducing reward utility functions $f_i : \mathbb{R} \to \mathbb{R}$. Then, for an agent $A \in \mathrm{Ags}$, their opponent $B \in \mathrm{Ags}$ and a path $\rho \in \mathrm{FPath}$ with $\mathrm{last}(\rho) = s$, *cognitive state utility function* $\mathtt{cu}^{\mathtt{s}}_{\mathrm{A}} : \mathrm{FPath} \to \mathbb{R}$ describes utility gained by A in s as follows:

$$\mathtt{cu}^{\mathtt{s}}_{\mathrm{A}}(\rho) = \sum_{i=1}^{k} \lambda_i^A f_i^A(\mathrm{r}^{\mathtt{s}}_{\mathrm{i,A}}(s)) + \sum_{i=1}^{l} \lambda_i^{A,A} f_i^{A,A}([\![\eta_i^A]\!]^\rho) + \sum_{i=1}^{l} \lambda_i^{A,B} f_i^{A,B}(\mathbb{E}_A^\rho[\eta_i^B]).$$

Hence, utility obtained by an agent in a state is a weighted sum of their physical rewards and mental rewards, where the latter are defined in terms of values of agent's own mental states and expected values of their opponent's mental states. Typically, some, if not most, of the mental goal coefficients $\lambda_i^{A,A}$ and $\lambda_i^{A,B}$ will be equal to 0 and the corresponding component of the utility function may be omitted, as the example below illustrates. Note also the the above definition applies when there are only two agents in the system, A and B. For every additional agent, another component corresponding to A's estimations of that agent's mental states must be added to A's cognitive utility function.

Example. In the tic-tac-toe example, parent's cognitive state utility at a path ρ whose last state is s is

$$\mathtt{cu}^{\mathtt{s}}_{\mathrm{p}}(\rho) = \lambda_1^p \mathrm{r}^{\mathtt{s}}_{1,\mathrm{p}}(s) + \lambda_1^{p,k} \mathbb{E}_p^\rho[\eta_1^k],$$

where, as before, η_1^k represents kid's satisfaction and state reward function $\mathrm{r}^{\mathsf{s}}_{1,\mathrm{p}}(s)$ assigns rewards for winning the game. On the other hand, child's utility lacks the cognitive component as they only care about winning the game:

$$\mathtt{cu}^{\mathsf{s}}_{\mathrm{k}}(\rho) = \lambda_1^k \mathrm{r}^{\mathsf{s}}_{1,\mathrm{k}}(s).$$

It may seem counter-intuitive that satisfaction does not feature in child's utility function, but our reasoning is that, during the game, the kid only thinks about winning, while satisfaction comes later, without the child fully realising the origin of that sensation (due to cognitive limitations). It illustrates the difference between the parent's long-term and the child's short-term thinking.

Things get a little more complicated when agent B attempts to compute their opponent's utility, i.e., the value of $\mathtt{cu}^{\mathsf{s}}_{\mathrm{A}}(\rho)$. First of all, B does not know the value of A's goal coefficients, maintaining a belief over their values instead. Second, B does not know the values of A's mental states and A's estimations of B's mental states. Using the expectation operator liberally, we may express the expected value of A's utility in the last state of some path $\delta \in \mathrm{FPath}$, computed by B after executing path ρ as

$$\begin{aligned}\mathbb{E}^\rho_B[\mathtt{cu}^{\mathsf{s}}_{\mathrm{A}}(\delta)] = \sum_{i=1}^{k} \mathbb{E}^\rho_B[\lambda_i^A] f_i^A(\mathrm{r}^{\mathsf{s}}_{\mathrm{i,A}}(s)) + \sum_{i=1}^{l} \mathbb{E}^\rho_B[\lambda_i^{A,A}] f_i^{A,A}(\mathbb{E}^\delta_B[\eta_i^A]) \\ + \sum_{i=1}^{l} \mathbb{E}^\rho_B[\lambda_i^{A,B}] f_i^{A,B}(\mathbb{E}^\rho_B[\mathbb{E}^\delta_A[\eta_i^B]]).\end{aligned}$$

Note two different paths present in the above expression – here we require that δ extends ρ (i.e., ρ is a prefix of δ), reflecting the fact that B computes *future* utility of A as part of their decision-making process (see below).

While B's expectations of A's goal coefficients are easily computed with respect to B's belief and $\mathbb{E}^\delta_B[\eta_i^A]$ is computed with respect to B's mental state estimation function, nested expectation is more difficult to resolve. In fact, rather than introducing a complex structure such as nested estimation, we assume agents approximate nested expectation, such as $\mathbb{E}^\rho_B[\mathbb{E}^\delta_A[\eta_i^B]]$, by taking the true value of their mental state at ρ, $[\![\eta_i^B]\!]^\rho$, and using mental state dynamics from there.

Formally, letting $\delta = s_0 a_0 \dots s_n a_n s_{n+1}$, nested expectation is resolved as follows:

$$\mathbb{E}^\rho_B[\mathbb{E}^\delta_A[\eta_i^B]] = \begin{cases} [\![\eta_i^B]\!]^\rho & \text{if } \rho = \delta, \\ \delta_i(\delta[0\dots n], a_n, A, B)(\mathbb{E}^\rho_B[\mathbb{E}^{\delta[0\dots n]}_A[\eta_i^B]]) & \text{otherwise.}\end{cases}$$

5.4.2 Decision Making Equations

Recall that, in order to select an action according to the softmax choice rule, an agent computes expected future utility corresponding to each available action. This computation, in general, is recursive in nature, where the depth of the recursion is given by an agent's lookahead parameter β. The recursion explores the game tree, computing expected utilities at each visited state, including states where the agent's opponents take actions. To predict actions of the other player, an agent computes its expectation of their opponent's future utility, which, combined with the softmax choice rule, allows an agent to quantify the likelihood of different future paths.

To formally express this expected future utility for agent A, we define a family of mutually recursive random variables $U^{\rho,\delta,h}_{A,A}$, $U^{\rho,\delta,h}_{A,B}$, $U^{\rho,\delta,h,a}_{A,A}$, $U^{\rho,\delta,h,a}_{A,B}$ and $P^{\rho,\delta,h,a}_{A,B}$ on the space Θ_B of B's meta-parameters. The first four of those random variables express different variants of expected (with respect to B's mental characteristics) utility of A or B (depending on the second subscript), accumulated over the next h steps, starting in the last state of δ, computed by A (as denoted by the first subscript) at a point of execution where current path is ρ. If an action a is specified in the superscript, the random variable denotes expected utility assuming a is taken after executing δ; otherwise, it is the expected utility in a state reached after executing path δ. The last random variable, $P^{\rho,\delta,h,a}_{A,B}$, expresses the probability that agent B takes an action a in the last state of δ, computed by A after executing ρ.

We present the definitions of those random variables below. We call this set of equations *decision making equations.* To improve readability, we fix an element $\langle \alpha_B, \beta_B, \gamma_B \rangle \in \Theta_B$ and implicitly assume that all occurrences of the random variables below are passed this element as an argument. We also assume that the current path is ρ and δ is a finite future extension of ρ, with $\mathrm{last}(\delta) = s$. The definitions are as follows:

$$U^{\rho,\delta,h}_{A,A} = \begin{cases} \mathsf{cu}^{\mathsf{s}}_{\mathrm{A}}(\delta) & \text{if } h = 0, \\ \mathsf{cu}^{\mathsf{s}}_{\mathrm{A}}(\delta) + \sum_a P^{\rho,\delta,h,a}_{A,\textsc{owns}(s)} U^{\rho,\delta,h,a}_{A,A} & \text{if } h > 0. \end{cases} \tag{5.1}$$

$$U^{\rho,\delta,h}_{A,B} = \begin{cases} \mathbb{E}^{\rho}_{A}[\mathsf{cu}^{\mathsf{s}}_{\mathrm{B}}(\delta)] & \text{if } h = 0, \\ \mathbb{E}^{\rho}_{A}[\mathsf{cu}^{\mathsf{s}}_{\mathrm{B}}(\delta)] + \sum_a P^{\rho,\delta,h,a}_{A,\textsc{owns}(s)} U^{\rho,\delta,h,a}_{A,B} & \text{if } h > 0. \end{cases} \tag{5.2}$$

$$U^{\rho,\delta,h,a}_{A,A} = \mathsf{cu}^{\mathsf{a}}_{\mathrm{A}}(\delta, a) + \gamma_A \sum_{s'} \mathrm{T}(s,a)(s') U^{\rho,\delta a s',h-1}_{A,A} \tag{5.3}$$

$$U^{\rho,\delta,h,a}_{A,B} = \mathbb{E}^{\rho}_{A}[\mathsf{cu}^{\mathsf{a}}_{\mathrm{B}}(\delta, a)] + \gamma_B \sum_{s'} \mathrm{T}(s,a)(s') U^{\rho,\delta a s',h-1}_{A,B} \tag{5.4}$$

$$P_{A,B}^{\rho,\delta,h,a} = \frac{\exp(\alpha_B U_{A,B}^{\rho,\delta,h,a})}{\sum_{a' \in \textsc{actions}(s)} \exp(\alpha_B U_{A,B}^{\rho,\delta,h,a'})} \tag{5.5}$$

Hence, looking first at Equations 5.1 and 5.2, expected future utility computed by an agent depends on the time horizon. When it is equal to 0, utility may be computed directly; in Equation 5.1 it is simply an agent's own cognitive state utility, while in Equation 5.2 it is an expectation, computed with respect to one's own belief, of opponent's cognitive state utility. Otherwise, expected utility accumulated within the next h steps is equal to the cognitive state utility (or, again, an expectation of opponent's utility) gained in the current state plus a sum of expected utilities corresponding to different actions available to an agent in the current state (whichever agent owns that state), weighted by the probabilities of taking those actions. That probability, in turn, is computed according to Equation 5.5, where the expected utilities are expressed by random variables U.

Next, looking at Equations 5.3 and 5.4, expected utility corresponding to an action a taken in a state s (where $s = \text{last}(\delta)$) is simply action utility (in Equation 5.3, or an expectation of opponent's action utility in Equation 5.4), plus a discounted sum of expected utilities in states reachable from s, weighted by transition probabilities.

Finally, the probability of B taking an action a (Equation 5.5) is based on the softmax choice formula, adapted to the current notation. Note that we allow $A = B$, which represents an agent computing their own action.

With that, we can express the probability of agent A taking action a when the current path is ρ, computed by the system's modeller, as $\text{Prob}^{\rho}(a) = \mathbb{E}^{\rho}[P_{A,A}^{\rho,\rho,\beta_A,a}]$, where $A = \textsc{owns}(\text{last}(\rho))$. Similarly, the probability of agent B taking action a in the last state of δ, computed by agent A after executing path ρ, is simply an expectation (computed with respect to A's meta-parameter estimations of B) of the value of random variable $P_{B,A}^{\rho,\delta,\beta_B,a}$, formally expressed as $\text{Prob}_A^{\rho}(a) = \mathbb{E}_A^{\rho}[P_{B,A}^{\rho,\delta,\beta_B,a}]$.

Example. We briefly overview how the parent computes their action in the state $s_0 = [(1,1)]$ from Figure 5.2a. For simplicity, we assume $\beta_p = 3$ (short lookahead), $\alpha_p \to \infty$ (perfect rationality) and $\gamma_p = 1$ (no discounting); moreover, we assume the parent knows child's meta-parameters: $\beta_k = 2$, $\alpha_k = 5$ and $\gamma_k = 0.7$. We set parent's goal coefficients to $\langle 0.1, 0.9 \rangle$ – child's satisfaction is the parent's priority.

To decide on an action, an agent generally computes expected utilities corresponding to each available action (Equation 5.5) and chooses probabilistically – the higher the expected utility, the higher the chances of an action being picked. In a special case of perfect rationality, only actions that yield maximal expected utility are considered (usually there is only one such). The six actions available to

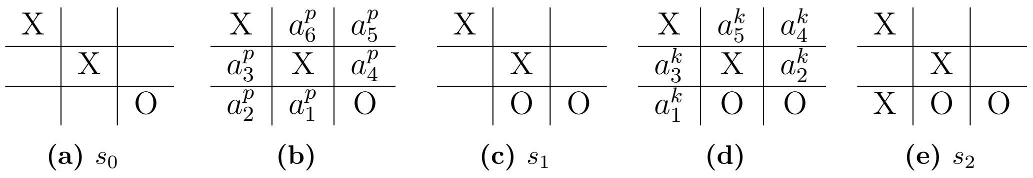


Figure 5.2: Reference for illustration of the decision-making process

Table 5.1: Expected utilities and action likelihoods computed by the parent

(a)

	a_1^k	a_2^k	a_3^k	a_4^k	a_5^k
$U_{p,p}^{\rho,\delta_1,2,\cdot}$	1.26	-0.17	-0.17	1.09	1.09
$P_{p,k}^{\rho,\delta_1,2,\cdot}$	0.74	0.06	0.06	0.06	0.06

(b)

	a_1^p	a_2^p	a_3^p	a_4^p	a_5^p	a_6^p
$U_{p,p}^{\rho,\rho,3,\cdot}$	1.05	0.04	1.05	0.5	0.04	0.5

the parent in s_0 are depicted in Figure 5.2b; Table 5.1b shows expected utilities corresponding to each action, computed according to decision making equations. We focus on a_1^p; in particular, letting ρ be the path taken to reach s_0, we analyse how $U_{p,p}^{\rho,\rho,3,a_1^p}$ is computed.

In the absence of action rewards, expected utility for a_1^p reduces to the expected utility in a state reached after taking action a_1^p (by Equation 5.3, given lack of discounting) – call it s_1 and the path taken to reach it δ_1. Now, in s_1, it is the child who takes an action, so computing $U_{p,p}^{\rho,\delta_1,2}$ requires the parent to quantify the likelihood of the child taking different actions (by Equation 5.1). Possible actions for the kid in s_1 are depicted in Figure 5.2d; to compute the probabilities of the child taking each of them ($P_{p,k}^{\rho,\delta_1,2,\cdot}$), the parent computes expected utilities of the kid corresponding to each action, according to Equation 5.4 (it involves using belief and meta-parameter estimations). It turns out that a_1^k is deemed most likely by the parent; specifically, $P_{p,k}^{\rho,\delta_1,2,a_1^k} \approx 0.74$ (recall that the kid is not perfectly rational, so they choose their action noisily), which is reassuring, since any other move loses the game for the kid (assuming optimal play on the parent's side).

All that remains for the parent is to compute their expected utility corresponding to each action of the child – the values are summarised in Table 5.1a. They are not hard to compute using Equation 5.3 – in the absence of action rewards, each $U_{p,p}^{\rho,\delta,2,a_i^k}$ is simply a sum of rewards in the next two states, the first of which is given by the transition function, while the second is easily determined by the parent.

With that, the value of $U_{p,p}^{\rho,\rho,3,a_1^p} = U_{p,p}^{\rho,\delta_1,2}$ (1.05) is given (according to Equation 5.2) by the sum of expected utilities weighted by probabilities from Table 5.1a.

5.4.3 Belief Update

We now formally describe the mechanism by which agents update their beliefs. Note that this section contains the abstract, conceptual treatment, while Section 6.1.5 goes into the details of how belief update is implemented.

Abstractly, agents update their belief in a Bayesian manner, upon observing their opponent's action. In other words, posterior belief is given as a product of prior belief and the likelihood of their opponent taking the observed action. The key aspect of this Bayesian update is the way the likelihood of a given action is computed. Suppose that at some point of the system execution the current path is ρ and it is A's turn to take an action, one of $a_1, a_2, \ldots, a_p$. Intuitively, based on their belief about A's characteristics, B assesses which actions are more likely than others. Then, upon observing the action taken by A, B's belief will be refined to reflect newly gained information. If the observed action had been deemed likely by B in the first place, their belief will be reinforced. If, on the other hand, B had not been expecting to see the action chosen by A, their belief may need to change substantially.

Formally, the likelihood of A taking action a, computed by B at a point of execution where current path is ρ, given a set of A's goal coefficients $\vec{\lambda}$ is expressed using the random variable P defined in Equation 5.5 as follows:

$$\mathrm{Prob}_B^{\rho,\vec{\lambda}}(a) = \mathbb{E}_B^{\rho}[P_{B,A}^{\rho,\rho,\beta_A,a} | \vec{\lambda}^A = \vec{\lambda}].$$

Note that the conditional expectation is used somewhat informally here; the expectation is computed with respect to B's estimation of A's meta-parameters, while the condition $\vec{\lambda}^A = \vec{\lambda}$ holds on a space Λ_A of A's goal coefficients. However, computing that expectation generally involves evaluating expressions of the form $\mathbb{E}_B^{\rho}[\mathsf{cu}_{\mathrm{A}}^{\mathsf{s}}(\delta)]$, which in turn necessitates computing B's expectations of A's goal coefficients. Normally, these would be computed with respect to B's belief, but, in this case, A's goal coefficients are conditionally given as $\vec{\lambda}$.

Now, supposing that A has taken action a_i (for some $i \in \{1, 2, \ldots, p\}$) and the system transitioned to a state s, B's belief is updated as follows:

$$\mathrm{be}_{\mathrm{B}}^{\rho s}(\vec{\lambda}) = \mathrm{be}_{\mathrm{B}}^{\rho}(\vec{\lambda}) \frac{\mathrm{Prob}_{B,A}^{\rho,\vec{\lambda}}(a_i)}{\int_{\Lambda_A} \mathrm{Prob}_{B,A}^{\rho,\vec{\lambda}'}(a_i) d\vec{\lambda}'} \tag{5.6}$$

Intuitively, having observed A take an action a_i, B infers that mental characteristics which maximise the likelihood of A taking action a_i are more probable than others.

5.4.4 Trust and Trustworthiness

As outlined above, a key contribution of our framework is the inclusion of agents' emotions in their decision-making process. In principle, any mental state can be expressed in our model, as long as two mechanisms are specified: (i) an evaluation function, which captures the nature of the emotion by setting out how an agent experiences it at any point of execution (we assume it can be represented numerically), and (ii) a dynamics function, which captures the heuristic agents use to estimate the value of this mental state in others, which they cannot observe.

A mental state we are particularly interested in as part of this work is trust – a complex mental attitude, generally understood as a subjective belief of a trustor towards a trustee about an action relevant to some goal (see Section 3.1 for an overview). In this section, we use trust as an example to illustrate the process of modelling an emotion in CSMGs. We begin with a brief overview of assumptions we make about the exact formulation of trust, followed by a formal definition in the setting of CSMGs (which serves as a trust evaluation function) and trust dynamics.

In this thesis, while we support the view that trust is task dependent, we assume that it has a universal component, which we call *core trust*. Inspired by social and cognitive science research [45, 59, 113], we define it as *a subjective belief of a trustor over trustee's willingness to go out of one's way to help the trustor achieve their goal*, and measured on a $0 - 1$ scale. Core trust between two agents A and B may be thought of as a measure of their mutual relationship – we would expect core trust between A and B to be high if they are good friends or family relatives and low if they had interacted previously with a disappointing outcome.

The concept of trust is very closely related to the notion of trustworthiness. Intuitively, while the value of trust is based on subjective belief of a trustor, trustworthiness is attributed to the trustee and is based on their mental attributes, past experience or preferences. We expect trust to change during repeated interactions, while trustworthiness remains constant. In fact, we follow Russell Hardin [113] in reducing trust to trustworthiness via agent's belief.

Each agent is assumed to have a baseline core trust, which measures one's propensity to trust, i.e., the initial level of core trust between oneself and a complete stranger. It is important to note that baseline core trust in most humans is a positive number, reflecting a degree of blind faith in others [114]. Another property of core trust is asymmetricity – A's trust towards B will in general differ from B's trust towards A, although the two will often be positively correlated. Finally, regarding the relationship between core trust and task-specific trust, we assume that the former is necessary, but not sufficient, for the latter. In particular, we take it that task-specific trust $\kappa_{A,B}(\psi)$ consists of core trust $\delta_{A,B}$ combined with task competence $\gamma_{A,B}$ via the following relationship

$$\kappa_{A,B}(\psi) = \delta_{A,B} \prod_{a \in \text{actions}(\psi)} \gamma_{A,B}(a),$$

where actions(ψ) are actions required to achieve task ψ. The representation of task ψ and the computation of actions needed to achieve ψ is application-specific and we do not cover it extensively at present. However, as suggested by the notation we used to refer to a task, a logical language such as PRTL* (introduced in Chapter 4) could be used for defining tasks.

Similarly, the way competence trust $\gamma_{A,B} : \text{Act} \to [0,1]$ is specified will differ between applications. We envisage an experience-based model, where previous performance of B on actions equal or similar to a, observed by A, affects the value of $\gamma_{A,B}(a)$. Endowing A with some initial trust in B's competence allows one to use Bayesian methods to update A's competence trust towards B, based on observations. However, at present we do not concern ourselves with the competence model and consider scenarios in which actions taken by agents do not require skill. Therefore, below we use terms *trust* and *core trust* interchangeably.

Before presenting a definition of core trust, we make a note about notation and nomenclature, intended to facilitate the reader's comprehension. Trust is a mental state, represented by a mental (latent) variable, e.g., $\texttt{trust}_{A,B}$ for A's core trust towards B. The actual trust that A feels towards B (known only to A), when execution history is ρ, is expressed as $[\![\texttt{trust}_{A,B}]\!]^{\rho}$ and computed according to the definition of trust given below. On the other hand, B will generally not know how much A trusts them, i.e., the value of $\texttt{trust}_{A,B}$, and will use the trust dynamics model (also given below), along with their initial estimation of A's trust, to compute their expectation of A's trust, denoted $\mathbb{E}^{\rho}_{B}[\texttt{trust}_{A,B}]$, at path ρ.

Core Trust

To formally define core trust, we restrict our attention to two agents, A and B, and focus on A's trust towards B, represented by $\texttt{trust}_{A,B}$. We note that A has a $\lambda^{A}_{\tau} f^{A}_{\tau}(\mathbb{E}^{\rho}_{A}[\texttt{trust}_{B,A}])$ component in their utility function, reflecting A's desire to be trusted by B. Intuitively, every agent is to some extent (or not at all, when $\lambda_{\tau} = 0$) motivated by maximising the value of core trust that their opponent places upon them.

Our observation is that trust between A and B is a measure of their relationship, and a desire of A to develop or maintain a relationship with B is equivalent to A

making an effort to increase or preserve B's trust. This desire is usually motivated by an expectation of future reward associated with such relationship, or a gratitude related to past events. In any case, the more A cares about their relationship with B, the more trustworthy A should appear to B. In fact, A's commitment to building a relationship with B (in other words, A's trustworthiness towards B), expressed as a numerical value between 0 and 1, is equal to λ_τ^A.

Hence, we define the *trustworthiness* of an agent as follows.

Definition 1. Let $\mathcal{M}$ be a cognitive stochastic multiplayer game, A and B be agents in $\mathcal{M}$, and let $\lambda_\tau^{A,B}$ be the goal coefficient of A corresponding to mental reward associated to $\texttt{trust}_{B,A}$ mental variable. Then A's trustworthiness (towards B) is given by

$$\tau_{A,B} = \lambda_\tau^{A,B},$$

i.e., it is equal to the value of the goal coefficient.

A noteworthy consequence of this definition is that we cannot assign a single trustworthiness score to an agent; it is only meaningful to talk of an agent's trustworthiness towards some other agent. This also explains why different people may have different opinions about someone's trustworthiness.

An important question that arises is the interpretation of a numeric value of trustworthiness. Recall that the parameter $\lambda_\tau^{A,B}$ measures relative importance that A assigns to maintaining a good relationship with B when making decisions. Indeed, when $\lambda_\tau^{A,B} = 0$, A is not concerned with their relationship with B at all; correspondingly, A's trustworthiness towards B is zero. Conversely, when $\lambda_\tau^{A,B} = 1$, A is solely motivated by strengthening their relationship with B, interpreted as the maximum trustworthiness with value one. However, trustworthiness values between 0 and 1 are less clear-cut. In fact, due to potentially varying numerical ranges of reward structures and the reward utility functions f_i, intermediate values of $\lambda_\tau^{A,B}$ can only be meaningfully interpreted given more context.

We are now ready to formally define core trust, which, recall, will be formulated as A's assessment (belief) of B's trustworthiness towards A. In particular, for each pair of agents A and B in a CSMG $\mathcal{M}$, we introduce a random variable $T_{A,B} : \Lambda_A \to [0,1]$ on the space Λ_A of A's goal coefficients, given by $T_{A,B}(\vec{\lambda}) = \lambda_\tau^{A,B}$. Hence, given a vector of goal coefficients of A, $T_{A,B}$ extracts trustworthiness of A towards B. With that,

Definition 2. Let A, B be agents in a CSMG $\mathcal{M}$ and ρ be the current path. Then, A's core trust towards B, denoted $\delta^{\rho}_{A,B}$, is defined as

$$\delta^{\rho}_{A,B} = \mathbb{E}^{\rho}_{A}[T_{B,A}],$$

where the expectation is computed with respect to A's belief.

Note an important property of this definition – since trust is generally viewed as a dynamic notion, a good formulation of trust should model this variation. Indeed, due to the way agents' beliefs are updated, an agent's trust also changes as a result of the opponent's action. It is important to realise, however, that, while the value of trust changes continuously, trustworthiness of agents remains constant (due to our assumption of goal coefficients not changing).

Trust Dynamics

Having introduced the formal definition of core trust, we now describe the heuristics that agents use to estimate how much their opponent trusts them. Before we do that, we briefly mention why this heuristics is needed. An alternative approach would involve an agent using the formal definition to compute their opponent's trust toward themselves. Since core trust of B towards A is defined in terms of B's belief, which in general is unknown to A, A would need to maintain a belief over B's belief in order to compute the expectation of B's trust. But then, since B generally needs to compute their expectation of A's utility in order to compute their own action, B would need to maintain a belief over A's belief over B's belief. When reasoning over longer paths, this nesting grows ever deeper and is not maintainable in practice. The trust dynamics model breaks this nesting and provides an efficient, yet realistic, mechanism for trust estimation.

Before proceeding, we note that a trust dynamics model is generally model-specific, as it is defined relative to actions available in a system. However, based on basic properties of trust, we provide below a set of principles and a general form of a trust dynamics model, which may be adapted to particular applications.

Normally, the first step of formulating a trust dynamics model involves identifying actions that require or affect trust. In many models, this can be accomplished using common sense, but alternative, more mechanistic approaches exist. One such is proposed by Wagner et al. [40], who formulate trust as a "belief, held by the trustor, that the trustee will act in a manner that mitigates the trustor's risk in a situation in which the trustor has put its outcomes at risk", and based on that definition, put forward four conditions necessary for trust considerations to arise

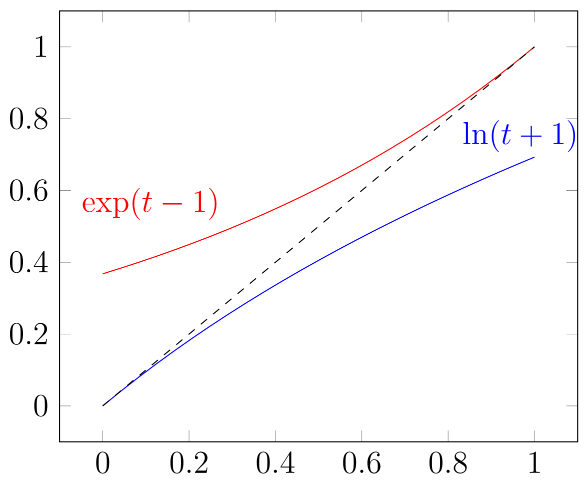


Figure 5.3: Curves suggested to model trust dynamics, depicted against the identity function

in a given setting. These can be verified on a game-theoretic representation of an analysed scenario, which integrates well with our framework.

Once conditions for trust are verified to hold, actions that put an agent at risk are identified (so-called *trusting actions*, using terminology from [40]), followed by finding actions of their opponent that maintain or violate trust placed on them.

Having identified the trust-related actions, each of them is then mapped to the effect it has on trust. The *maintain trust* and *violate trust* actions increase and decrease trustor's trust, respectively. We suggest functions $f(x) = \ln(x + 1)$ and $g(x) = \exp(x - 1)$ to represent the decrease and increase of trust, respectively. They are simple to compute, take on values in the interval $[0, 1]$ (for $x \in [0, 1]$), are increasing and have a desired property of approaching $h(x) = x$ at extremes (as seen in Figure 5.3). This ensures that trust increases slowly when it is high and decreases slowly when it is low.

Of course, not every action that maintains trust is equal and, in some cases, the distinction between maintaining and violating trust may be difficult to pinpoint. In such cases, the logarithm and exponential functions must be adapted to the unique requirements posed by a given system. In fact, in Section 5.5 we show exactly how that can be done, as part of modelling the trust game as a CSMG.

To sum up, we take task-specific trust to consist of two components: (i) task-independent *core trust*, which measures the willingness of an agent to go out of one's way to carry out its commitment to another, and (ii) task-dependent *competence trust*, which reflects the ability of an agent to carry our a given task and is based on past performance and circumstances at the time. We assume A's core trust towards B is an estimation (by A) of B's trustworthiness, which in turn is defined with respect to B's goal coefficients.

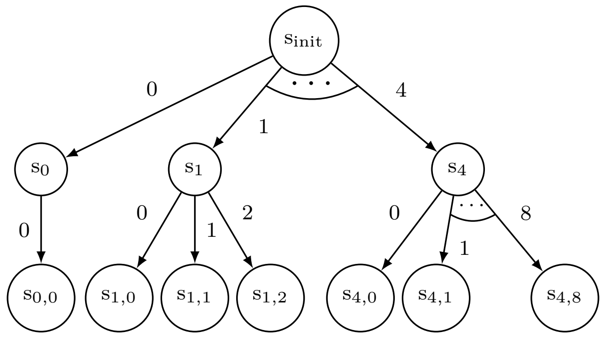


Figure 5.4: One iteration of the trust game

5.5 The Trust Game Example

Similarly as for ASMASs, we put the theory into practice by considering the trust game example. This serves not only as a demonstration of our framework, but also as a comparison of CSMGs with ASMASs.

First thing to note is that we consider a different version of the trust game than in Section 4.4. It is a variant much more commonly used in the literature. It still involves two agents, Alice and Bob, who enter a money-exchange scenario. As before, Alice is endowed with \$10. However, instead of a binary choice between investing and withholding, she can now share any integral part of her endowment with Bob. Whatever amount she invests gets duplicated in transit before being received by Bob, who can share any (integral) part of it with Alice. Note that Bob receives no endowment in this variant. Figure 5.4 shows a graphical representation of the game where the initial endowment of Alice is \$4 (rather than \$10) to help readability. Note that states are subscripted with the sequence of actions taken up to that point in the game. Also, in the figure, actions are represented as integers, but we also denote them as a_0 (action of transferring \$0), a_1 (action of transferring \$1) and so on, whenever it helps readability.

As mentioned in Section 4.4, standard game-theoretic analysis based on equilibria predicts that no cooperation will arise in this scenario. However, a crucial assumption needed to arrive at that conclusion is that players only care about money; then Bob has no incentive to share any of his profits and Alice, anticipating Bob's behaviour, has no reason to invest with him. This reasoning may be applied iteratively, which means that no cooperation is anticipated even in a repeated version of the trust game. However, human experiments show that non-zero transfers occur in the majority of interactions [54], contradicting the theoretical predictions. Evidently, human decision making is not as simple as maximising monetary payoffs.

Indeed, we postulate that personality of the agents and the trust between them should be included in the analysis. Humans are social animals (an observation that goes back to Aristotle) guided by social norms, such as reciprocity, and driven by a desire to develop new, and preserve existing, interpersonal relationships. Or, they may only care about the money, but use their awareness of social norms to take advantage of more reciprocative humans. Therefore, trust will be modelled as a mental state in our formalism, allowing us to capture how trust considerations shape decision making of agents.

5.5.1 Game Setup

The standard components of the model are similar to those of the binary variant of the game, but reflect the changes in game semantics.

First of all, the state of the game no longer contains the cognitive components (goals and intentions of agents), but, to allow easy generalisation to the repeated trust game, it now records all past investments and past returns of agents, along with turn information[1]. For example,

$$\begin{aligned} s_{\text{init}} &= (\text{Alice}, [], []), \\ s_1 &= (\text{Bob}, [a_1], []), \\ s_{1,2} &= (\text{Alice}, [a_1], [a_2]). \end{aligned}$$

Other possible states include $s_{3,5,3,4} = (\text{Alice}, [a_3, a_3], [a_5, a_4])$ or $s_{3,1,4,1,2} = (\text{Bob}, [a_3, a_4, a_2], [a_1, a_1])$. In general, for a state s_{seq} the subscript *seq* of s is a sequence of actions taken by agents so far. Importantly, since states encode execution histories, we may abuse the notation and use states whenever paths are required[2]. In particular, expressions such as $\mathtt{cu}^{\mathtt{s}}_{\text{A}}(s_{1,2})$, $\mathtt{cu}^{\mathtt{a}}_{\text{A}}(s_{3,5,4,4}, a_2)$ or $\mathbb{E}^{s_{2,2}}_{A}[\mathtt{trust}_{\text{Alice,Bob}}]$ will be treated as shorthand notation for corresponding expressions where states are replaced by paths starting in s_{init} and ending in that state.

Transition function is formally given by

$$\mathrm{T}((\mathit{turn}, \mathit{invs}, \mathit{rets}), a) = \begin{cases} (\text{Alice}, \mathit{invs}, a : \mathit{rets}) & \text{if } \mathit{turn} = \text{Bob}, \\ (\text{Bob}, a : \mathit{invs}, \mathit{rets}) & \text{if } \mathit{turn} = \text{Alice}, \end{cases}$$

where the notation $a : \mathit{rets}$ denotes a list obtained by prepending an action a to a list of past returns *rets*.

[1]We note that recording turn information is redundant, but helps readability and facilitates notation.

[2]This is important to keep in mind, especially when relating formulas in this section to the formal semantics.

Physical Rewards

A single physical reward structure $\mathrm{R} = \{\mathrm{r}_{\mathrm{Alice}}, \mathrm{r}_{\mathrm{Bob}}\}$ represents monetary incomes of agents, where state and action rewards are defined in a natural way (formally specified below).

$$\mathrm{r}^{\mathbf{s}}_{\mathrm{Alice}}(s) = \begin{cases} 4 & \text{if } s.\mathit{turn} = \text{Alice}, \\ 0 & \text{otherwise.} \end{cases}$$

$$\mathrm{r}^{\mathbf{s}}_{\mathrm{Bob}}(s) = 0 \text{ for all } s.$$

$$\mathrm{r}^{\mathbf{a}}_{\mathrm{Alice}}(s, a) = \begin{cases} -a & \text{if } s.\mathit{turn} = \text{Alice}, \\ a & \text{otherwise.} \end{cases}$$

$$\mathrm{r}^{\mathbf{a}}_{\mathrm{Bob}}(s, a) = \begin{cases} 2a & \text{if } s.\mathit{turn} = \text{Alice}, \\ -a & \text{otherwise.} \end{cases}$$

Therefore, initial endowments take the form of state rewards in state s_{init}. Transfers, on the other hand, are naturally modelled as action rewards, with Alice's transfer duplicated in transit.

Mental Rewards

Having described the standard components of our model of the trust game, we now move on to the novel elements, starting with mental rewards. As mentioned above, we are interested in capturing the trust between agents, represented by two latent variables, $\texttt{trust}_{\mathrm{Alice,Bob}}$ and $\texttt{trust}_{\mathrm{Bob,Alice}}$. Recall that a mental reward structure consists of the mental variables themselves as well as an evaluation function and a dynamics model for each mental state. In this case, we only have one mental variable for each agent and the evaluation function is based on our definition of trust from Section 5.4.4, which formulates it as an expectation of another's trustworthiness, computed with respect to agent's belief. Hence, for an agent $A \in \mathrm{Ags}$ and an execution history ρ

$$\omega_{\tau}(\rho, A) = \mathbb{E}^{\rho}_{A}[\lambda^{B,A}_{\tau}]$$

gives the value of A's trust towards B, which only A knows.

It is the trust dynamics model that requires more effort. Recall that in Section 5.4.4 we have presented a generic model of trust evolution and a basic recipe for constructing the dynamics function for a given model. Below, we show how to apply that procedure to the trust game.

Trust Dynamics The first step involves identifying actions that give rise to trust considerations. It is clear that Alice's decision to share any part of her endowment puts her at Bob's mercy, i.e., makes her vulnerable. Her choice is a classic example of an action that requires trust – by investing, she stands a chance to earn more than she would had she not invested, but she also risks losing part of what she was given. Assuming she invests, and Bob cooperates, her trust is rewarded and, in most cases, increases. On the other hand, if Bob keeps most of the profit to himself, Alice will likely regret her decision and lose trust in Bob.

Conversely, taking Bob's point of view, depending on how much he trusts Alice, he may have different expectations of her behaviour. Lack of cooperation (i.e., low or nil investment) from Alice's side denies Bob an opportunity to make a profit, and hence we may expect Bob's trust to decrease. On the other hand, if Alice is unexpectedly generous in her investment, Bob will likely gain trust towards her.

However, trust considerations are further complicated by the wider range of transfers available to agents (compared with the binary variant). Our trust dynamics model must differentiate between low and high transfers – e.g., upon an investment of \$6 by Alice, a return of \$4 by Bob will likely produce a different trust response than a return of \$7.

Finally, we note that trust evolution is not uniform with respect to time – initially, agents have little knowledge about each other, so trust changes resulting from behavioural observations are greater. In later rounds of the game, the value of trust becomes more established and each subsequent action of an opponent comes with less influence on trust. The evidence supporting this property has been observed experimentally [35, 74].

In what follows, we assume that the multiplication factor of the game is 2, i.e., the amount Alice sends to Bob gets duplicated in transit. Generalisations to other values, in particular $k = 3$, are possible, but require small modification to the methods presented below. On the other hand, the presented procedures are independent of the size of Alice's endowment.

The high-level structure of the update procedure is outlined in Algorithm 14. It describes how one agent's trust changes as a response to their opponent's action. It captures the non-uniformity of trust dynamics by decreasing the computed trust change by a factor which increases as the game progresses. The action taken by one of the agents might be an investment or a return, which, as described above, have different nature and are treated separately by the algorithm.

Algorithm 14: Trust dynamics function

```
input  : prior trust $t_0$, action $a$, execution history $\rho$
output : updated value of trust $t$
function updateTrust($t_0$, $a$, $\rho$):
    $n \leftarrow$ current round number;    /* 1,2,3,... computed using $\rho$ */
    $d \leftarrow \sqrt{1/n}$;
    /* determine if $a$ is an investment using last state of $\rho$ */
    if $a$ is an investment then
        $t \leftarrow$ trustChangeAfterInvestment($t_0$, $a$, $\rho$) ;    /* See Alg 15 */
    else
        $inv \leftarrow$ most recent investment on $\rho$;
        $t \leftarrow$ trustChangeAfterReturn($t_0$, $inv$, $a$) ; /* See Algorithm 17 */
    end
    return $t_0 + d(t - t_0)$ ; /* discount updates as game progresses */
end
```

Algorithm 15: Dynamics of trust – update following an investment

```
input  : old (prior) trust $t_0$ of Bob towards Alice, Alice's investment $inv$,
         Alice's endowment $endow$, game history $\rho$
output : updated trust of Bob towards Alice (perceived by Alice)
function trustChangeAfterInvestment($t_0$, $inv$, $\rho$):
    $p \leftarrow inv$ as a proportion of Alice's endowment;
    $t_1 \leftarrow$ expCurve($p$, $t_0$) ;    /* See Figure 5.5 */
    $nice_{\text{Bob}} \leftarrow$ computeBobNiceness($\rho$);
    $nice_{\text{Alice}} \leftarrow$ computeAliceNiceness($inv$, $\rho$);
    $t_2 \leftarrow$ trustChangeNiceness($nice_{\text{Bob}}$, $nice_{\text{Alice}}$, $t_0$) ;    /* Alg 16 */
    $w \leftarrow$ significance of niceness ;    /* See Figure 5.8 */
    return $t_1(1 - w) + t_2 w$;
end
```

Trust Change Following an Investment The meat of trust dynamics is encoded in functions *trustChangeAfterInvestment* and *trustChangeAfterReturn*; we begin with an overview of the former, outlined in Algorithm 15. It captures the heuristics Alice uses to update her estimation of Bob's trust upon her investment. Note that the procedure operates with Alice's endowment as input, rather than assuming the usual value of \$10. The basic idea behind the update is that high investments increase Bob's trust while low investments decrease it. However, the amount invested must be judged relative to the value of Bob's trust prior to the investment (t_0). If it was low, then even a moderate investment (half of endowment, say) will likely increase it (but a higher investment would increase it more).

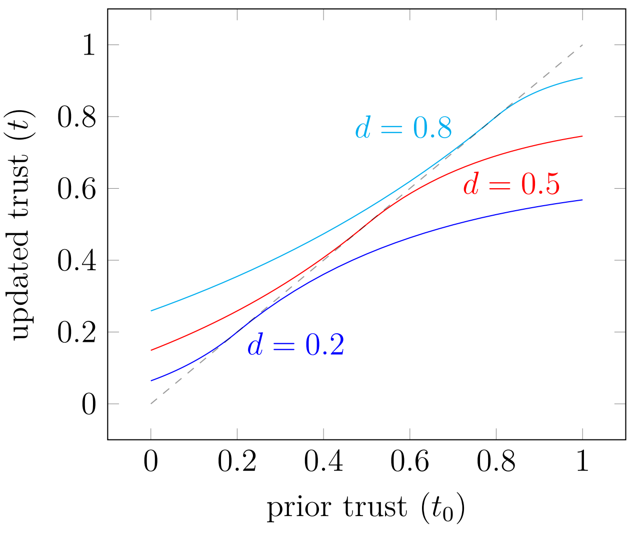


Figure 5.5: "Exponential" trust curves; each consists of two distinct functions of the form $a \exp(bx) + c$ that cross identity ($f(x) = x$) at $x = d$; the parameters for each section are chosen so that the first approaches identity from above on $[0, d]$ (representing trust increase that declines as prior trust grows) while the second deviates from identity on $[d, 1]$ (representing trust decrease that gets bigger as prior trust grows)

The second caveat is that the way Bob's trust evolves will depend on his past behaviour, particularly his returns in the most recent rounds of the game (if any). For example, if in the previous round Bob returned more than he received, he may expect Alice to increase her investment accordingly.

Hence, we postulate that the new value of trust is a weighted average of two estimates: one computed based on the relationship between Bob's prior trust and the size of Alice's investment and another based on the relationship between how nice Bob has been so far in the game and how nice Alice's current investment was.

The first of those estimates, denoted t_1 in Algorithm 15, is computed using a function designed to capture the relationship between Alice's investment and Bob's prior trust. Note that the investment is expressed as a proportion of maximal possible investment (which is equal to Alice's endowment) so that it takes values between 0 and 1. Figure 5.5 depicts the curves that model how trust evolves upon an investment. The intuition is that Bob's expectation of Alice's investment corresponds to his trust towards her. For example, if his trust is 0.7, he expects Alice to invest roughly 70% of her endowment. If she invests more, his trust will increase and if she invests less, his trust will decrease. Hence, given an investment proportion, the trust curves cross the $y = x$ identity line when x is equal to that proportion.

The second estimate, denoted t_2 in the algorithm, is computed based on comparing Alice's and Bob's "niceness" – a number between -1 and 1 that summarises an agent's past behaviour in the game. For Alice, niceness captures

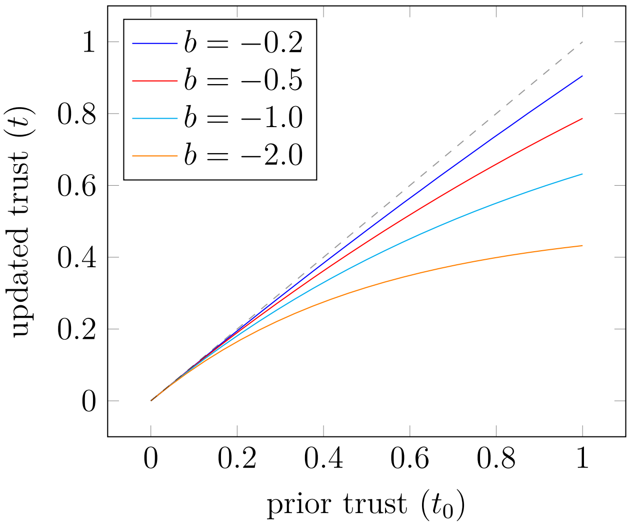


Figure 5.6: Exponential curves $f(x) = ae^{bx} + c$ with $f(0) = 0$, for selected values of b

the relationship between her most recent investment and the one before – it is negative when most recent investment is lower and positive when it is higher than the one prior to that. For Bob, niceness is more complex – it aggregates all his past behaviour. In each round, his one-step niceness expresses the proportion of Alice's investment that he returned, shifted towards -1 to obtain the required range. With that, overall niceness is a weighted average of one-step niceness over all rounds, with more recent behaviour assigned more importance using exponential smoothing.

Algorithm 16: How trust changes depending on niceness

```
input  : prior trust $t_0$ of Bob towards Alice, Bob's niceness $nice_{\text{Bob}}$, Alice's
         niceness $nice_{\text{Alice}}$
output : updated trust of Bob towards Alice (perceived by Alice)
1 function trustChangeBasedOnNiceness($t_0$, $nice_{Bob}$, $nice_{Alice}$):
2     if $nice_{Alice} < 0$ and $nice_{Bob} > 0$ then
3         $b \leftarrow (nice_{\text{Bob}} - nice_{\text{Alice}})\ /\ 4$;
4         return expCurveCrossingAtZero($-b$, $t_0$);
5     else
6         $b \leftarrow nice_{\text{Alice}} > nice_{\text{Bob}}\ ?\ nice_{\text{Alice}} - nice_{\text{Bob}} : 0$ ;  /* Ternary op */
7         return expCurveCrossingAtOne($b$, $t_0$);
8     end
9 end
```

Then, the value of t_2 is computed based on the relationship between Alice's and Bob's niceness – the way it is done is outlined in Algorithm 16. If Bob has been nice but Alice has not (the if branch of the if statement), Bob's trust decreases, with the magnitude of the decrease proportional to the difference in niceness. This

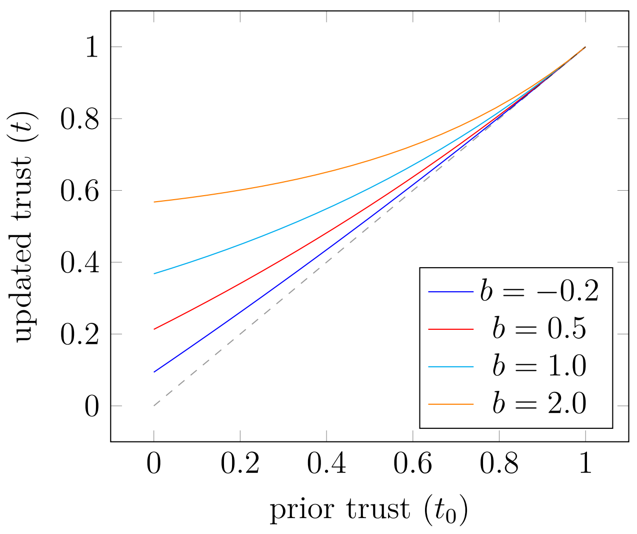


Figure 5.7: Exponential curves $f(x) = ae^{bx} + c$ with $f(1) = 1$, for selected values of b

dynamics is achieved by using exponential functions of varying steepness that cross the origin, illustrated in Figure 5.6. Otherwise, if Alice was nicer than Bob (the positive case of the ternary operator of else branch), his trust will increase according to a function from a family illustrated in Figure 5.7 – again, larger differences in niceness correspond to less steep curves, i.e., greater increases in trust. Finally, if Bob was nicer than Alice but both were nice or both were not nice (the negative case of the ternary operator of else branch), we assume the trust does not change (when $b = 0$ the exponential curve is assumed to coincide with the identity function).

With t_1 and t_2 computed, the last aspect of the algorithm involves determining the weights of the two trust values when they are averaged. As indicated on line 7 of Algorithm 15, the weights are based on how significant Bob's niceness is deemed to be. This significance is computed based on two factors: (i) the number of rounds played so far in the game – more rounds means more evidence; (ii) the value of Bob's niceness itself – more extreme values are assumed to be more significant. The resulting relationship between the number of past rounds, Bob's niceness and the weight of t_2 is depicted in Figure 5.8.

Trust Change Following a Return The hypothesised evolution of trust following Bob's return of Alice's investment, outlined in Algorithm 17, is much simpler than what we have just described for investment. It is modelled using exponential functions from Figure 5.5 – the central part of the algorithm involves computing the point at which the curve will cross $y = x$ line. The important thing to note is that this point does not get uniformly closer to one as the return proportion

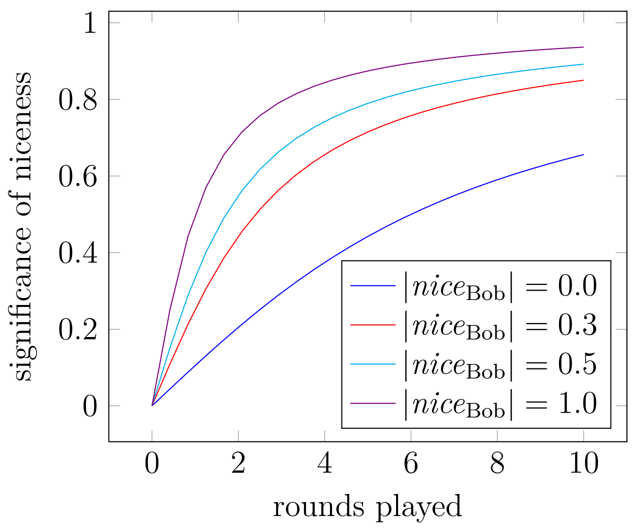


Figure 5.8: How weight of t_2 depends on Bob's niceness and the number of rounds played so far

Algorithm 17: Dynamics of trust - update following a return

```
input  : old trust $t_0$ of Bob towards Alice, Alice's investment $inv$, Bob's
         return $ret$
output : updated trust of Alice towards Bob (perceived by Bob)
function trustChangeAfterReturn($t_0$, $inv$, $ret$):
    $p \leftarrow ret\ /\ inv$;
    if $p > 1$ then
        $c \leftarrow 1 - (1 - t_0)^{\frac{ret-(inv+1)}{inv}}$;
    else
        $c \leftarrow (9p/10)^2$;
    end
    return expCurve($c$, $t_0$) ;                    /* See Figure 5.5 */
end
```

increases. Instead, returns that are close to twice the investment are in fact deemed untrustworthy, unless the investment was low and the value of trust was low.

This design captures the following intuition: when Alice's trust is low and she invests a small part of her endowment ($1 or $2, say), then Bob's high return should be interpreted as an encouraging sign – he shows willingness to cooperate without significantly sacrificing profits (since investment was low anyway). However, assuming Alice's trust is already high and she invests a larger part of her endowment ($6 or $7, say), then Bob's high return may be considered suspicious, as he puts himself at a monetary disadvantage for no apparent reason. For example, it may be that Bob is a con man, intending to quickly gain Alice's trust, thereby encouraging large investments, and subsequently defect, keeping the money to himself.

Hence, we now give the trust dynamics function $\delta_\tau : \mathrm{FPath} \times \mathrm{Act} \times \mathrm{Ags} \times \mathrm{Ags} \to ([-1,1] \to [-1,1])$. Given agents A and B, action a taken in the last state of some path ρ and estimated prior trust t_0 (before a was taken) of B towards A, $\delta_\tau(\rho, a, A, B)(t_0)$ describes how A's estimation of B's trust changes following action a being taken:

$$\delta_\tau(\rho, a, A, B)(t_0) = \begin{cases} \mathit{updateTrust}(t_0, a, \rho) & \text{if } \textsc{owns}(\mathrm{last}(\rho)) = A, \\ t_0 & \text{otherwise.} \end{cases} \tag{5.7}$$

Hence, A's estimation of B's trust changes when A takes an action and it is updated according to Algorithm 14.

Utility Function We are now almost ready to specify an agent's utility functions. The only components that remain to be given are the reward utility functions. For monetary income, represented by the reward structure R, we use an identity function – hence utility gained from earning \$4 is equal to 4. Note that this is not generally true, as utility associated to monetary outcomes and incomes have been shown to follow an S-shaped curve which is steeper for losses than for gains [98]. However, since the amounts we consider as part of the trust game are small, linear utility is an acceptable approximation.

On the other hand, utility of trust is assumed to be described by a function $f_\tau(t) = 2Et$, where E is Alice's endowment. The linear coefficient is chosen to correspond to a maximal possible earning of any player during one round of trust game, so that the utility gained from monetary income is comparable in magnitude with the utility gained from relationship-building. This in turn ensures that goal coefficients of an agent are comparable in value, rather than strongly biased. We emphasise, however, that this seemingly arbitrary choice of the linear coefficient of the trust utility function does not fundamentally alter the workings of the model – it should rather be thought of as a stylistic choice.

Therefore, given a path ρ whose last state is s, cognitive state utility of agents gained at s is given by:

$$\mathtt{cu}^{\mathbf{s}}_{\mathrm{Alice}}(\rho) = \lambda_m^{\mathrm{Alice}} \mathrm{r}^{\mathbf{s}}_{\mathrm{Alice}}(s) + \lambda_\tau^{\mathrm{Alice}} 2E\mathbb{E}^{s}_{\mathrm{Alice}}[\mathtt{trust}_{\mathrm{Bob,Alice}}],$$
$$\mathtt{cu}^{\mathbf{s}}_{\mathrm{Bob}}(\rho) = \lambda_m^{\mathrm{Bob}} \mathrm{r}^{\mathbf{s}}_{\mathrm{Bob}}(s) + \lambda_\tau^{\mathrm{Bob}} 2E\mathbb{E}^{s}_{\mathrm{Bob}}[\mathtt{trust}_{\mathrm{Alice,Bob}}].$$

Additionally, utility gained by agents when action a is taken in state s is described by:

$$\mathrm{u}^{\mathbf{a}}_{\mathrm{Alice}}(s, a) = \lambda_m^{\mathrm{Alice}} \mathrm{r}^{\mathbf{a}}_{\mathrm{Alice}}(s, a),$$
$$\mathrm{u}^{\mathbf{a}}_{\mathrm{Bob}}(s, a) = \lambda_m^{\mathrm{Bob}} \mathrm{r}^{\mathbf{a}}_{\mathrm{Bob}}(s, a).$$

That completes the specification of the trust game as a CSMG. In Section 6.2.1, we use the model just defined to make behavioural predictions. In Chapter 7, we describe an experimental study with human subjects in which we used our model to drive the behaviour of an artificial bot.

5.6 Conclusions

We have presented an emotion-aware model of human decision making that served as a setting for a novel definition of social trust. Our framework is data-driven and captures the intricacies of human reasoning using insights from theory of mind and existing behavioural models. A wide range of human personalities may be expressed in the framework thanks to a continuous representation of each agent's characteristics via a vector of goal coefficients. We have shown how to model the trust game using our framework, which involved formalising the evolution of trust in humans. The rest of this thesis illustrates the applications of our model.

6

Cognitive Stochastic Multiplayer Games – Implementation & Case Studies

Contents

This chapter covers the practical side of cognitive stochastic multiplayer games – we begin by overviewing its probabilistic programming implementation in Section 6.1, followed by presentation of several case studies that illustrate the depth of applications of our framework, in Section 6.2

6.1 Implementation

Recall that one of the main goals when creating our framework was implementability – we envisage it being used in robots, enabling them to smoothly integrate in the society. This section describes the tool that we have developed to that end. Besides showing that our original goal has been achieved, an implementation allows us to validate and showcase our model more effectively.

The first decision one faces when preparing to implement a theoretical model is that of which programming language, or, more generally, programming paradigm, to use. We have opted for WebPPL [79], a recently developed probabilistic programming language embedded in JavaScript. This decision has been influenced by a recent rise in the quality and popularity of probabilistic programming languages. Indeed, Ong et al. [23] point out several advantages of expressing models of emotions as stochastic programs and argue for more widespread use of probabilistic programming in affective computing. These mental theories expressed as probabilistic programs are highly compatible with our framework and could be used as submodules of our tool, as long as it is developed in a language that supports stochasticity.

Recent trends aside, many aspects of our framework, such as agents' beliefs, estimations or their decision-making process, are inherently probabilistic. Probabilistic programming languages provide constructs for sampling from a variety of probabilistic distributions and making inferences out of the box, thereby greatly simplifying the development process.

Having settled on the paradigm, one must then select a particular language. As mentioned above, we have chosen WebPPL – its ease of use, plenty of support available online and existence of the related code base [115, 116] being the main reasons behind our selection. In particular, WebPPL allows one to express all major probability distributions and provides an easy mechanism for sampling from them. Inference is equally easy to perform and supports conditioning. Furthermore, WebPPL has already been used to implement models of rational agents in MDPs [115]. Our tool can be viewed as a continuation of that line of work – we take the basic model they proposed and extend it with mental reasoning.

6.1.1 Overview

We now give an overview of the tool[1], which serves also as a summary of the operation of our framework. Recall that our main contribution is a novel formalisation of agent decision making. Its most basic application is an ability to *simulate* the execution of a turn-based, stochastic game, which constitutes a primary feature of our tool. Formally speaking, the tool computes the posterior predictive distribution over future actions of agents. It requires several inputs: mechanics of the game (states, actions, transition function etc.), specification of the mental component (mental states, their

[1]Available at `https://github.com/maciekolejnik/webppl-cognitive-agents`

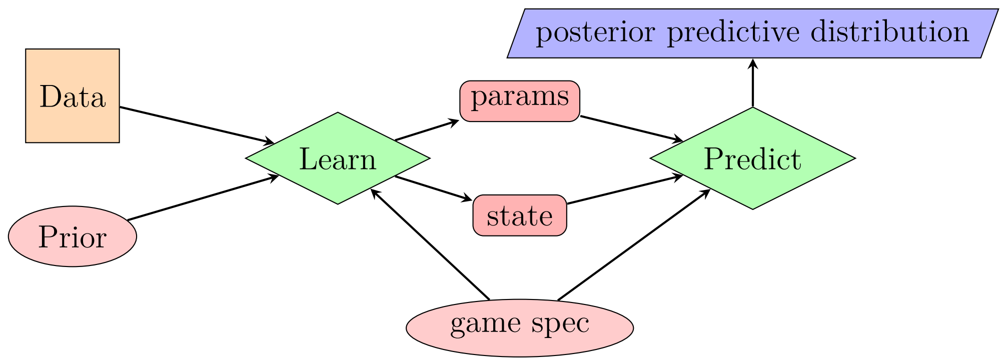


Figure 6.1: Overall process of the framework

dynamics and evaluation functions) and each agent's utility function, parameters and initial state. The simulation may then be run for any number of iterations of the game and it generates a sequence of actions taken at each step. Note that this is a probabilistic process (actions are sampled from the computed posterior predictive distribution) and the computed trace will generally differ between executions; what remains the same, however, are the probability distributions according to which agents choose their actions. As the execution progresses, players update their beliefs in a Bayesian fashion (see Section 6.1.5 for how that is implemented), assimilating the information gained by observing their opponents' actions.

The main obstacle encountered when simulating execution of a game is the need to provide parameters and initial states of agents. These are often not known; even if some information about the agents is at hand, it may not be easy to translate it to the required format. However, behavioural data, i.e., a record of actions taken by agents, is likely to be much more readily available. Hence, the second functionality of our tool involves learning preferences, beliefs and estimations of agents from data. The inference is again Bayesian in nature and uses the decision-making model to compute the likelihoods of various actions. It also requires a prior, which may encode population-wide statistics, our belief about characteristics of a particular agent or be set to uniform if no information is available. The process is graphically summarised in Figure 6.1.

6.1.2 Representations

Recall that each agent in our framework maintains (and updates) their belief $\mathrm{be}^{\rho}_{\mathrm{A}}$, as well as meta-parameters estimations θ_A and mental state estimations $\mathrm{est}^{\rho}_{\mathrm{A}}$. Abstractly, all those components can be arbitrary probability distributions; however, their implementation requires finite representations that allow efficient sampling and, in case of belief and mental estimations, updating.

Table 6.1: Distributions used for meta-parameter estimations

rationality α	Gaussian
lookahead β	Poisson
discount factor γ	Beta/Dirac

Belief

Recall that belief is defined on the space $\Lambda = \{\vec{\lambda} \mid \sum_i \lambda_i = 1\}$ of possible goal coefficient vectors. As it turns out, family of Dirichlet distributions, overviewed in Section 3.2.3 and commonly used in Bayesian statistics, is highly appropriate for expressing structures of this type. This yields a compact representation of belief as a vector $\vec{\alpha}$ of positive reals. The length of $\vec{\alpha}$ depends on the number of agents in the system and the format of agents' utility functions; however, letting n denote the number of agents, k be the number of physical rewards and l be the number of mental variables, the length of $\vec{\alpha}$ is no greater than $k + nl$.

Meta-parameter Estimations

When it comes to agents' estimations of others' meta-parameters, our choice is less constrained due to the assumption that agents do not update them. Of course, we still need a finite representation, but in this case our choice boils down to selecting a distribution that is most suitable for a given parameter. Our selections are summarised in Table 6.1 and briefly overviewed below. Note that in practice, continuous distributions are often replaced by their discrete approximations, which makes inference by enumeration possible.

Rationality Recall that, for an agent A, α_A is a positive real number that measures A's rationality, roughly interpreted as an ability of an agent to correctly assess utilities of available actions and select the best one. It is well-known that qualities of this type follow a Gaussian distribution within a given population.

Lookahead Next, lookahead β of an agent describes how far into the future they look when considering their next move. The range of possible values of β is the set of non-negative integers and we expect the distribution of β in a population to approximately follow the familiar bell curve. That can be modelled with a Poisson distribution without significantly compromising accuracy. It has an added benefit of simplicity, with the unique parameter λ that may be set to the most likely value of β for a given agent.

Discount Factor Finally, the discount factor γ of an agent takes values between 0 and 1, which makes Beta distribution suitable to represent its estimation. However, in some applications agents may be known to not discount future rewards, in which case a Dirac distribution may be used. Another possibility that calls for a δ-distribution is when any value of the discount factor that is not 1 is equivalent from the point of view of agents' decision making. An example of such a system is our model of the game of tic-tac-toe analysed in Chapter 5.

Mental State Estimations

Recall that agents update their estimations of mental states of others using dynamics functions that operate on scalars. This update mechanism operates pointwise on the support of the estimation. Hence, we take mental state estimations to be represented using categorical distributions, ensuring finiteness of the support and termination of the update mechanism.

6.1.3 Inference

Recall from Section 3.2.4 that exact probabilistic inference is often intractable, particularly when the distributions involved are continuous. As a result, approximate methods, such as sequential or Markov chain Monte Carlo, are often employed. In the context of our tool, using continuous probability distributions to represent meta-parameter estimations would necessitate utilising these inexact techniques. However, due to the high complexity of the decision-making mechanism and inefficiency of WebPPL, approximate algorithms do not perform well within our tool. We hypothesise that this stems from the fact that these methods rely on visiting many points in the probability space. However, computing likelihood of each point, which involves invoking the decision-making algorithm, is an expensive operation in our tool. As a result, the approximate inference algorithms do not terminate within reasonable time bounds. Instead, we use exact inference throughout the tool, which is made possible by making sure that all continuous probability distributions are discretised, as indicated in Section 6.1.2.

6.1.4 Code Highlights

To give the reader a better idea of how our tool is implemented, we now discuss its internal structure and present selected code fragments. The description in this section is largely conceptual, but it is complemented by the user guide for the tool, which can be found in Appendix A.

Recall that WebPPL is a functional subset of JavaScript; hence, our tool is written in a functional style. The bulk of the most relevant code is contained in two functions: (i) `makeCSMG`, which takes a basic setup of a game (states, actions, transitions, physical and mental rewards of agents) as an argument and returns an object representing a CSMG, and (ii) `makeAgent`, which instantiates an agent, given its parameters, its initial state (belief, mental estimations and meta-parameter estimations) and a game (obtained by calling `makeCSMG`) it is part of.

Making a Game Out of these two functions, `makeCSMG` is much simpler – besides validating provided inputs, a non-trivial task in itself, its main job involves converting the supplied dynamics models into a mental reward structure. The key code fragment that captures the complexity of computing mental rewards is displayed in Listing 5. First of all, note that, in our program, this function is defined inside the scope of various other methods (such as `makeCSMG`), which explains why most of the variables may appear undefined.

In any case, the crucial aspect of `computeRewardsFromAttitude` is that identifiers of three agents are introduced, reflecting the number of parties involved in the computation of mental rewards. First, we have the agent who computes the value of (a mental component of) a utility function, `selfAgentID`. Second, we have the agent whose utility is being computed, `ofAgentID` (of course, it may be that `selfAgentID` = `ofAgentID`). Third, there is the agent whose mental state is being estimated (or computed), `overAgentID`.

The utility function, whose form `selfAgentID` is assumed to know, will generally contain various components corresponding to different mental attitudes. In this case, we are focusing on one such attitude, identified by `mentalAttIdx` – let us assume it is trust. Now, `ofAgentID` may care about many other agents' trust, including their own trust. All those agents are listed in `agentArr`, which is a parameter that comes from the game setup provided by the user as argument to `makeCSMG`. As the listing shows, computing rewards involves iterating (i.e., mapping, as we are in the realm of functional programming) over `agentArr`, and, for each `overAgentID` whose trust must be computed, invoking `estimateReward`.

There are three possibilities:

1. `selfAgentID` computes its own utility and its own trust.
   In this case, computation of the reward boils down to using the definition of trust, passed as a property of the model and encoded in the `computeMentalState` function.

```
/** This function captures how an agent (identified by
* *selfAgentID*) computes mental rewards associated to a given
* mental attitude (identified by *mentalAttIdx*), gained by
* *ofAgentID* from *overAgentID*'s mental state.
* This computation is performed in *state*, where *selfAgentID*'s
* belief is *belief*; *mentalSnapshot* provides a reference
* point for computing nested estimation; *agentArr* is an array
* of agent IDs, identifying agents whose mental state *ofAgentID*
* cares about. */
let computeRewardsFromAttitude = function(mentalAttIdx, agentArr) {
  let estimateReward = function(overAgentID) {
    if (selfAgentID === overAgentID) {
      if (selfAgentID === ofAgentID)
        return computeMentalState(state, belief, mentalAttIdx)
      else
        return nestedEstimation(state, mentalAttIdx, mentalSnapshot)
    }
    return expectation(
      estimateMentalState(state, overAgentID, mentalAttIdx)
    )
  }
  return map(estimateReward, agentArr)
}
```

Listing 5: Computing mental rewards

2. `selfAgentID` computes another agent's utility, but the component being computed relates to `selfAgentID`'s trust.
   This is an example of a nested estimation, which is resolved by combining the definition of trust and trust dynamics. In particular, `selfAgentID` uses the actual value of its trust towards `ofAgentID` at the present moment, but applies trust dynamics from there (as described in Section 5.4.1), reflecting the fact that the rewards are typically being computed for some future state.

3. `selfAgentID` computes its own or another agent's utility, and the component being computed relates to another agent's trust.
   In this case, mental state estimation function is employed, i.e., the trust dynamics model is used to estimate `overAgentID`'s trust.

Making an Agent The other function, `makeAgent`, encodes the decision-making process of an agent, the way its utility is computed and the way its belief is updated. Below, we describe in some detail the first of these mechanisms; utility computation uses the goal coefficients provided as an argument to `makeAgent`, but most of its complexity is in resolving mental rewards, outlined above; belief update is covered in Section 6.1.5.

Three functions are involved in the implementation of agents' decision making. Firstly, `act` (see Listing 6) kicks off the process by saving an agent's mental snapshot and encoding the aggregation over meta-parameters of their opponents. Then, for each combination of opponents' meta-parameters, the decision tree is explored using mutually recursive `actRec` (see Listing 7) and `expectedUtility` (see Listing 8) functions.

```
let act = function (state) {
  let turn = turn(state)
  // prepare mentalSnapshot for future utility computations
  let belief = belief(state)
  let mentalState = mapN(function(i) {
    return computeMentalState(state, belief, i)
  }, numberOfRewards.mental)
  let mentalSnapshot = {
    values: mentalState,
    state: state
  }
  let actionDist = Infer({method: 'enumerate'}, function () {
    let othersMetaParams =
      sampleMetaParamsEstimations(metaParamsEstimations)
    let allMetaParams = mergeMetaParams(
        othersMetaParams, selfMetaParams, selfId)
    let lookAhead = allMetaParams.lookAhead[turn]
    let actionRecDist = actRec(
      state, lookAhead, allMetaParams, mentalSnapshot)
    return sample(actionRecDist)
  })
  return actionDist
}
```

Listing 6: Computing an action – initial call

Note that `act` captures the computation of an agent's own, as well as their opponent's, action. The mental snapshot, which records an agent's mental state, is used to compute nested mental state estimations as part of `computeRewardsFromAttitude` function outlined above. Then, the action distribution is computed by enumerating possible combinations of meta-parameters of this agent's opponents, which we assume there is a finite number of. Of course, in principle, meta-parameter estimations may take the form of a continuous probability distribution, such as those suggested in Section 6.1.2. However, we have found that approximate inference methods do not perform well in this context. We hypothesise this has to do with the fact that each execution of the stochastic model passed to `Infer` is a heavy computation that itself involves inference. For each combination of meta-parameters, lookahead of the decision-maker is retrieved and an exploration of the decision tree begins with a call to `actRec`.

This relatively simple function encodes the soft expected utility maximisation principle using the `factor` operator of WebPPL (line 10). `factor` is similar to `condition`, introduced in Section 3.2.4, but weaker; rather than completely discarding some executions, it merely makes some more, and others less, likely. In particular, `factor`(x), when executed as part of a stochastic model passed to `Infer`, increases the log probability of the current execution by x. As part of `actRec`, this has an effect of assigning probability proportional to $e^{\texttt{alpha * eu}}$ to an action whose expected utility is computed as `eu`, in line with Equation 3.1.

```
 1  let actRec =
 2    function (state, timeLeft, allMetaParams, mentalSnapshot) {
 3      let turn = turn(state)
 4      let alpha = allMetaParams.alpha[turn]
 5      let availableActions = actions(state)
 6      let actionDist = Infer({method: 'enumerate'}, function () {
 7        let action = uniformDraw(availableActions)
 8        let eu = expectedUtility(state, action,
 9          turn, timeLeft, allMetaParams, mentalSnapshot)
10        factor(alpha * eu)
11        return action
12      })
13      return actionDist
14  }
```

Listing 7: Computing an action – recursive call

It is important to note that the computation of an expected utility of taking an action, captured by the `expectedUtility` function, reflects the possible future developments, in line with agent's lookahead. Specifically, expected utility of an action is a sum of the utility associated to that action (line 5) and a discounted expectation of future utility (line 25). The latter is computed recursively (line 23) with respect to a distribution of future actions (line 19).

Note also that the horizon used when computing this future action (line 17) is not always equal to the lookahead of the agent whose turn it is – it will often be the case that the lookahead of the agent who computes the expected utility, or whatever remains of it (`timeLeft`), does not reach that far. As an example, consider an agent A with a lookahead of three, who, as part of calculating future utilities, is now computing an action of their opponent B in a state s that is two steps away. Even if B's lookahead is estimated to be three, A will not be able to look that far into the future, as its own lookahead only allows A to consider one more future step.

```
let expectedUtility = function (state, action, ofAgentID,
  timeLeft, allMetaParams, mentalSnapshot) {
    if (timeLeft === 0) return 0
    let u = actionUtility(
      state, action, ofAgentID, mentalSnapshot)
    let nextTimeLeft = timeLeft - 1
    let discountFactor = allMetaParams.discountFactor[ofAgentID]
    let futureUtilityDist =
      Infer({method: 'enumerate'}, function () {
        let nextState = sample(transitionFn(state, action))
        let nextStateUtility = stateUtility(
          nextState, ofAgentID, mentalSnapshot)
        if (nextTimeLeft == 0)
          return nextStateUtility
        let nextTurn = turn(nextState)
        let nextHorizon = min(
          allMetaParams.lookAhead[nextTurn], nextTimeLeft)
        let nextActionDist = actRec(nextState, nextHorizon,
          allMetaParams, mentalSnapshot)
        let nextAction = sample(nextActionDist)
        return nextStateUtility + expectedUtility(
          nextState, nextAction, ofAgentID,
            nextTimeLeft, allMetaParams, mentalSnapshot)
    })
    let futureUtilExp = expectation(futureUtilityDist)
    let eu = u + discountFactor * futureUtilExp
    return eu
}
```

Listing 8: Computing an action – expected utility

Running Simulations The code described so far can be considered as the core of our tool, as it implements the theoretical assumptions behind our framework. However, another significant portion of our codebase consists of utility functions that facilitate usage of our tool: running simulations and performing inferences.

Of particular interest is the first of these functionalities. We provide an easy, yet flexible way of simulating executions of systems through a notion of *experiments* that consist of *scenarios*. The idea is that an experiment represents some aspect of a modelled interaction that we want to study, while scenarios define various conditions under which we want to run the model. With that, users do not need to manually create a game and agents, followed by repeatedly computing actions to simulate system execution. Instead, they can use the functionality we provide to only specify aspects that uniquely reflect their requirements.

In particular, each experiment comes with a name, a description and a list of scenarios, along with an optional set of functions that allow arbitrary code to be executed during simulations (designed to facilitate reporting). In turn, each scenario defines the parameters and initial states of agents, and game parameters. We then provide a function, presented in a simplified form (with input validation removed) in Listing 9, which, if supplied with a specification of a system (`makeGame`), a list of experiments and a command, will process that command and execute it, which typically involves running one or all of the experiments provided. Appendix A provides more details about the expected format of the commands and gives examples.

```
let processCommandline = function(makeGame, experiments, argv) {
  let log = argv.log
  if (log) {
    globalStore.loggingLevel = log
  }
  let help = argv.help
  if (help) {
    printInfo(experiments)
    return
  }
  let experimentIDStr = argv.experiment
  if (isDefined(experimentIDStr)) {
    /** run only selected experiment */
    let experimentID = parseInt(experimentIDStr)
    let experiment = experiments[experimentID]
    let result = run(experiment, makeGame, argv)
    return extend(result, { experimentID })
  }
  return runAll(experiments, makeGame, argv)
}
```

Listing 9: Running experiments

The crucial piece of code in Listing 9 is line 16, which runs a given experiment. Its operation is similar to `processCommandline` itself, in that it processes the command further to check whether one or more scenarios should be run, and delegates the rest to a function responsible for running individual scenarios. The actual simulation is captured by a function called `trace`, presented in Listing 10. In the spirit or functional programming, it uses a recursive helper to compute subsequent actions of an agent for a specified number of steps. It returns a trace, which consists of a list of $(\text{state}, \text{action})$ pairs (represented as arrays), but whose last element is a single state. Incidentally, `trace` illustrates a peculiar feature of WebPPL, namely that functions stored as fields of an object must be retrieved into a variable before being applied. That explains why lines 7 and 8, or 10 and 11, which one may be tempted to combine into one, must stay as they are.

```
1  // s_0 is the starting state of the simulation
2  let trace = function (game, s_0, agents, timeLeft) {
3    let traceRec = function(state, timeLeft) {
4      if (timeLeft > 0) {
5        let turn = game.API.turn
6        let actingPlayer = agents[turn(state)]
7        let act = actingPlayer.act
8        let actionDist = act(state)
9        let action = sample(actionDist)
10       let transitionFn = game.transitionFn
11       let nextState = sample(transitionFn(state, action))
12       return [[state, action]].concat(traceRec(nextState,timeLeft - 1))
13     }
14     return [[state]]
15   }
16   traceRec(s_0, timeLeft)
17 }
```

Listing 10: Simulating execution of a system

Simulations feature heavily in the analysis of the case studies in Section 6.2, with details of how to run them and examples of experiments and scenarios to be found in Appendix B.

Making Inferences Finally, we mention the second functionality of our tool: learning parameters and state of agents from data. Conceptually, it is a simple process, which involves combining the prior, defined individually for each inference, reflecting what is known about the agents, with the likelihood, given by the `act` function that encodes agents' decision making, and the evidence, and using Bayesian update to obtain a posterior on agent's state and parameters. This operation is performed repeatedly, for every available piece of evidence, which in this case takes the form of agents' actions.

Evidence is provided as a data file specifying a set of agents, parameters of the game and a behavioural record, i.e., a sequence of actions taken by agents and states that the game visited, possibly over multiple executions of the system. We define a custom format that this file must conform to and a set of methods for parsing it.

Conceptually, the most complex aspect of the inference is the representation of the belief we are learning, since it is a probability distribution over agent's parameters and state, which includes mental state estimations that are themselves represented as probability distributions. To avoid cumbersome nested structures in our code, we reduce mental state estimations to their expectations, so that our belief over them admits a tractable representation. Fortunately, beliefs of agents are succinctly encoded as vectors of parameters, which ensures probability distributions over these beliefs are tractable.

The key element of this inference process, presented in Listing 11, is the implementation of the Bayesian update. It involves computing an action that an agent would select under various combinations of parameters and state sampled from the prior. The likelihood of the sample is then given by the function `actionSimilarity` (line 20), which quantifies how similar the computed action is to the one observed. By convention, this similarity measure takes nonpositive values, with equal actions assigned a similarity score of 0 and distinct actions a negative score that increases in magnitude the more dissimilar the actions are. By default, distinct actions are assigned a score of -1. However, in many scenarios, especially when actions have a numerical interpretation, a more fine-grained notion of similarity may be given. An example is provided by the trust game, where the similarity of two monetary transfers is inversely proportional to the absolute difference between them.

The tipping example of Section 6.2.3 illustrates how inference is performed in practice.

6.1.5 Belief Update

In this section, we describe how beliefs of agents are updated in practice. Recall that, in principle, the update is Bayesian in nature, computed based on likelihoods of available actions. However, since we have settled on representing belief using a Dirichlet distribution, we must ensure that this representation is preserved after belief

```
// below code is in the scope of various other functions
// that process the data file.
// It is called to learn from observing
// *observedAction* taken in *state* by agent identified
// by *actingAgentID* in a system *csmg*
let agentPosterior = Infer({method: 'enumerate'}, function() {
  let agentSetup = sample(prior[actingAgentID])
  let agentParams = {
    goalCoeffs: agentSetup.goalCoeffs,
    metaParams: agentSetup.metaParams
  }
  let agentInitialState = {
    belief: agentSetup.belief,
    mentalEstimations: agentSetup.mentalEstimations,
    metaParamsEstimations: agentSetup.metaParamsEstimations
  }
  let agent = makeAgent(agentParams, actingAgentID, agentInitialState, csmg)
  let act = agent.act
  let computedAction = sample(act(state))
  factor(actionSimilarity(state, computedAction, observedAction))
  return agentSetup
})
```

Listing 11: Updating the prior over agent's parameters and state based on observation

is updated. Unfortunately, performing the update according to Equation 5.6 does not generally guarantee that the distribution remains a member of the Dirichlet family.

To define a procedure that ensures that belief maintains its representation, we use the fact that the Dirichlet distribution is a conjugate prior to categorical distribution. In other words, when the prior is distributed as a Dirichlet and the likelihood is a categorical distribution, the posterior will also be distributed as a Dirichlet, and the update will be a straightforward operation on the parameters of the distribution.

Recall that the Dirichlet distribution is naturally applicable to experiments with k outcomes and it describes our current belief about the probability of each outcome. It is parameterised by k values $(\alpha_1, \dots, \alpha_k)$ where each α_i is interpreted as $\alpha_i - 1$ observations is category i. Therefore, a natural update of the parameters upon making a new observation is to increment by 1 the parameter corresponding to the category in which the observation was made.

As it happens, in our model, the likelihood is indeed a categorical distribution; however, its support is made up of actions available to an agent. On the other hand, belief of one agent over another is a distribution over goal coefficient vectors, where each element of such a vector can be thought of as a weight this agent assigns to the corresponding reward. Therefore, when belief is represented as $Dir(\vec{\alpha})$, each element of $\vec{\alpha}$ quantifies an agent's conviction that their opponent is motivated by the corresponding reward. To define belief update, we must associate an observed action to a weighted set of rewards. More formally, having observed an action a performed by agent B, agent A updates its belief by incrementing one or more of α_is by amounts proportional to the weight(s) of one or more reward structures that a is mapped to.

To define this mapping, consider first its inverse: at a given point of the system execution where it is B's turn to take an action, each component of the utility function may be associated with a distribution over actions available to B, representing probabilities that an agent would take each action if they were solely motivated by that reward. As an example, if B's utility function has two components: (i) monetary incomes and (ii) opponent's trust, then, at any point of the system execution, (i) will favour actions that provide an agent with the largest income, while (ii) will favour actions that increase opponent's trust. The upshot is that, upon observing B take an action a, another agent A may associate a list of pairs (λ_i, p) to a, one for each λ_i in B's utility function, where λ_i is a goal coefficient identifying a reward and p is a (hypothetical) probability, computed by A, that B would take action a if they were solely motivated by reward corresponding to λ_i. Then, each α_i in the vector representing A's belief is incremented proportionally to the corresponding probability, where the sum of increments is equal to 1. The procedure just described is formalised in Algorithm 18.

Hence, the size of increment of each parameter (identified by an integer i), is proportional to the probability that agent B takes action a when guided solely by the i^{th} reward. Intuitively, that favours rewards which are likely to have motivated B to take action a.

The function *actCond* in line 5 of the algorithm represents a standard computation of opponent's action commonly performed in our tool, modified by a stipulation that utility is computed with respect to the i^{th} reward only. We may express this computation using notation of decision making equations as follows:

$$\text{Prob}_A^{\rho,i}(a) = \mathbb{E}_A^{\rho}[P_{A,B}^{\rho,\rho,\beta_A,a} \mid \text{utilities computed wrt. } i^{th} \text{ reward only}],$$

where the conditional probability operator is again used liberally; it should be interpreted to mean that, when evaluating $P_{A,B}^{\rho,\rho,\beta_A,a}$, the utility of B is always computed as if it only consisted of the i^{th} reward.

Algorithm 18: Approximate belief update (under Dirichlet representation)

```
input  : prior belief $\vec{\alpha_0}$ (represented as Dirichlet), current path $\rho$, agent $A$
         who updates their belief, agent $B$, action $a$ taken by $B$
output : updated belief $\vec{\alpha}$
1  function updateBelief($\vec{\alpha_0}$, $\rho$, $A$, $B$, $a$):
2      $\vec{ps} \leftarrow []$;
3      $n \leftarrow$ number of components in $B$'s utility function;
       /* compute probability of $B$ taking action $a$ under each
          reward structure                                           */
4      foreach $i$ in $0 \rightarrow n$ do
           /* call to actCond represents $A$ computing a distribution
              over possible actions of $B$ under condition that only
              $i^{th}$ reward is used for computing utilities           */
5          $\delta \leftarrow$ actCond($A$,$B$,$\rho$,$i$);
           /* retrieve probability of action $a$ from computed
              distribution                                           */
6          $ps[i] \leftarrow \delta(a)$;
7      end
8      $total = \sum_i(ps)$;
9      $\vec{\alpha} = []$;
10     foreach $i$ in $0 \rightarrow n$ do
11         $\alpha[i] \leftarrow \alpha_0[i] + ps[i]/total$;
12     end
13     return $\vec{\alpha}$;
14 end
```

6.2 Case Studies

To illustrate the expressiveness of our framework and show how it can be used in practice, we now consider a variety of scenarios that call for mental reasoning in humans. In each case, we formulate a CSMG model of a given interaction and use our tool and synthetic data to perform a number of experiments involving making predictions and inferences. Appendix B provides instructions for how to replicate the results described below.

6.2.1 Trust Game

We begin with the familiar by now trust game example, which we have already modelled as a CSMG in Section 5.5. Below, we assume players' endowments to be 4 and 0 for an investor and an investee, respectively. As usual, we take the multiplication factor k to be equal to 2.

To validate our proposed setup, we use the tool to predict the behaviour of a collection of agents to see whether it matches our expectations. We then use synthetic data to learn agents' parameters and predict their future behaviour. We take the following list of hypotheses to guide our experiments:

(H1) There exists a configuration of agent parameters under which cooperation is predicted by the framework.

(H2) As initial trust between agents increases, so does the social welfare of the game.

(H3) Untrustworthy individuals who try to trick their opponent into trusting them are punished.

The first hypothesis (H1) serves as a sanity check for the framework – it must hold or our approach is deemed to fail. The second hypothesis (H2) aims to confirm a widely held conviction that high trust is beneficial to the society – in this case, measured by average size of a transfer in trust game. Finally, the third hypothesis (H3) asserts that fair, trustworthy agents can detect when their opponent attempts to take advantage of their good intentions to make monetary profit and they do not cooperate with such con men.

In the experiments described below, we focus on the way agents' characteristics and initial beliefs influence their behaviour. Therefore, we fix agents' meta-parameters by assuming they're almost perfectly rational ($\alpha = 100$), have a limited lookahead ($\beta = 2$) and discount future rounds ($\gamma = 0.8$). We furthermore assume that agents estimate their opponent's meta-parameters accurately. We also reduce initial mental estimations (i.e., estimations of opponent's trust) to their expectations.

Verifying Cooperation Arises To verify that H1 holds, it suffices to make both agents trusting (i.e., their beliefs should be biased towards λ_τ) and trustworthy (i.e., $\lambda_\tau > \lambda_m$). In particular, we set both agents' goal coefficient vectors to $[0.3, 0.7]$ (where, recall, 0.3 is the weight an agent assigns to money, while 0.7 measures importance of trust) and their initial belief to $\mathrm{Dir}([1, 3])$, which implies initial trust equal to 0.75. Initial trust estimations matter little from the point of view of generated prediction – we assume they are fairly accurate, represented with a Dirac distribution centred at 0.7. Simulating execution of such a system produces highly cooperative behaviour, with investor investing the maximal possible amount (\$4) every time and investee reciprocating with returns at least equal to that. Of course, simulation has a probabilistic nature, so the resulting trace is not guaranteed

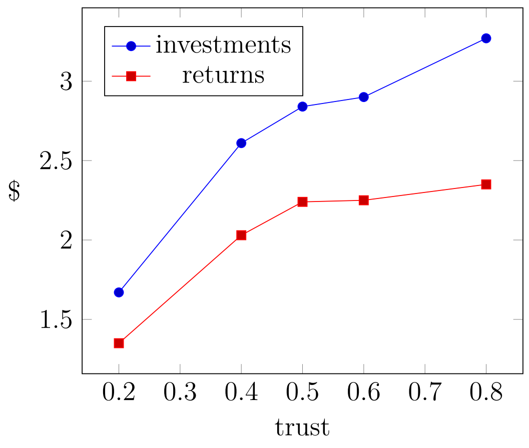


Figure 6.2: How mean (averaged over twenty simulations of ten executions of rounds of the trust game) investment and return depend on initial trust

to always be the same. However, with choice of parameters as specified above, most of the computed action distributions are very strongly biased towards one action, producing very predictable traces.

Investigating Effect of Trust Next, to test H2, we perform simulations of ten rounds of the trust game for various values of initial belief (and hence, initial trust), with other parameters chosen randomly. In particular, we use five different values of initial beliefs (Dir([4, 1]), Dir([3, 2]), Dir([2, 2]), Dir([2, 3]), Dir([1, 4])), which give rise to five different values of initial trust (0.2, 0.4, 0.5, 0.6 and 0.8). For each value, we define twenty scenarios, each with a combination of goal coefficients and trust estimations of agents that was randomly selected at the start of the experiment, and which is the same for the respective runs that vary in initial trust. In other words, the first (out of twenty) scenario for which initial trust is 0.2 will use the same parameters of agents as the first scenario for which initial trust is 0.4, 0.5 etc.

Moreover, to counteract randomness that arises from selecting actions according to a probability distribution as part of simulations, we perform six runs of each scenario. We are interested in mean investment and mean return corresponding to each value of initial trust, averaged over twenty scenarios and the six runs within each scenario. The results, plotted in Figure 6.2, confirm our hypothesis. We note a steep increase when initial trust increases from 0.2 to 0.4, suggesting that low values are particularly detrimental to social welfare. We also observe a widening gap between investments and returns – this stems from the fact that growing trust between agents generally leads to higher investments of Alice, but it does not necessarily cause Bob to be more generous.

Con Man Modelling Finally, to verify H3, we note that an untrustworthy individual is naturally modelled in the framework by setting $\lambda_m = 1$. However, a true con man (playing as Bob) might try to trick their opponent into trusting them by initially cooperating (particularly if initial investments are low) and then defecting (i.e., keeping all the profits for themselves), once investments become high enough. However, such a strategy can only profit the con man if he times his defection to coincide with the end of the game. To enable that, we assume he knows how many rounds the game will last. Moreover, we equip the con man with a more simplistic trust dynamics model – he assumes that, the higher his return, the greater the increase in Alice's trust towards him. Finally, the con man assumes that Alice's behaviour is driven by her trust – the more she trusts him, the more she will be willing to invest.

To see what happens when a cognitive agent faces a con man, we randomise Alice's goal coefficients, her belief and her initial estimation of Bob's trust and equip Bob with a high lookahead ($\beta_{Bob} = 5$) to model his strategic thinking. We then simulate the execution of the trust game for three rounds. Such a short horizon was chosen to play well with Bob's strategy and enable him to execute his tricks. Arguably, most con men do not engage with their victims for extended periods of time; rather, they act fast, attempting to gain someone's trust and quickly use it to their advantage.

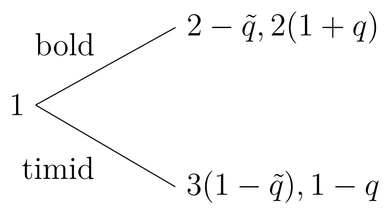
To obtain a wide range of Alice's personalities, we repeat this three-round simulation twenty times, ignoring the runs where no transfers arise (when Alice's trust is too low for her to invest). Preliminary results are mixed; we observed a wide variety of traces, reflecting diversity of Alice's parameters. Bob's strategy is fairly consistent throughout executions; he is generally noncooperative (i.e., does not return any money), unless an investment in the first or second round is low, in which case he senses an opportunity to trick Alice into trusting him and cooperates. However, he is most profitable when Alice's trustworthiness and initial trust are high. In that case, despite Bob's selfish behaviour, Alice's investments remain sizable throughout the game, as the rate of trust decline is gentle. Overall, this experiment illustrates that our framework is capable of expressing a wide variety of agent types. However, further investigation is needed to resolve our hypothesis.

6.2.2 Bravery Game

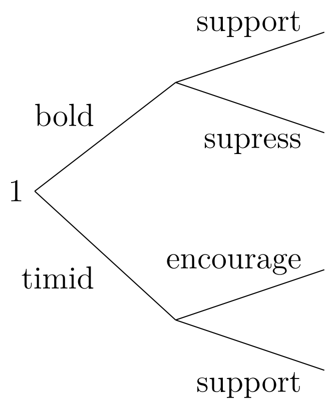
The second case study we analyse is inspired by an example from Geanakopolos et al [52] and serves as a comparison between psychological games (described in Section 2.2) and our model. The original scenario involves an agent (player 1) who makes a decision in front of his friends (player 2). He prefers to be *timid*, but he does not want to disappoint his friends, who may expect him to act *boldly*. Beliefs of players are captured by letting p be the probability of player 1 taking action *bold*, q be player 2's expectation of p (first-order belief) and $\tilde{q}$ be player 1's expectation of q (second-order belief). The game and agents' preferences are summarised in Figure 6.3a (in the style of the original paper); note that utility functions are defined in terms of beliefs, a crucial feature of psychological games framework. For example, the more player 1 thinks his friends expect him to be bold (reflected by a value of $\tilde{q}$ close to 1), the more likely he is to act boldly. Note that player 2 not only prefers their friend to act boldly, they moreover prefer to think of him as bold. Three distinct equilibria are predicted for this game: one in which player 1 acts boldly ($p = 1$), one in which he acts timidly ($p = 0$) and one where he randomises ($p = 0.5$). We note also that a crucial assumption that gives rise to these three equilibria is that beliefs of players correspond to reality in equilibrium, i.e., $p = q = \tilde{q}$ in each of the above solutions.

We propose to use the CSMG framework to conduct complementary analysis. We are interested in the expected behaviour of agents when their beliefs are *not* accurate and we wish to find out under what circumstances an equilibrium is reached and whether it is one of the equilibria predicted by the psychological games framework. We therefore consider a repeated version of the bravery game and, to enable player 1 making inferences about his friends' preferences, we modify the game to allow player 2 to react to player 1's decision. In particular, following a bold move, player 2 may *support* or *suppress* their friend's action, whereas, after a timid decision, possible reactions are *encouragement* or *support*. A graphical representation of (one iteration of) the resulting game is displayed in Figure 6.3b. Note that the notion of equilibrium is not defined in our framework, but we informally say that a simulation has reached an equilibrium if, from some point onward, action distribution computed by agents and their beliefs do not change from one iteration to the next.

Before giving an overview of a CSMG model of the bravery game, we outline the major differences in how mental states of agents are formalised in our framework and in psychological games. Belief plays an important role in both models but its scope differs: our belief is dynamic and computable and it quantifies uncertainty over opponent's characteristics; their (psychological games) belief operates on strategies



(a) As a psychological game



(b) As a CSMG

Figure 6.3: Bravery game

of other agents and does not support updating. Our model allows the formulation of rich, parameterised, belief-dependent mental states, whereas in psychological games emotions *are* beliefs. Moreover, while psychological games introduce nested beliefs of arbitrary order, our framework proposes a heuristic approach of mental estimations that simplifies beliefs of higher orders.

To represent the standard, numerical (non belief-based) component of agents' utility from Figure 6.3a, we define two physical reward structures: $\mathrm{R_b}$ assigns a unit reward to both players in a state reached after a *bold* move, while $\mathrm{R_t}$ assigns a unit reward after a *timid* move. With that, appropriately set goal coefficients allow us to model boldness or timidness of various degree. To represent belief-dependent components of agents' utility, we introduce a mental variable $\eta = pride$ and are particularly interested in $pride_{p_2}$, pride felt by player 2 upon player 1 making his decision. Note that we allow negative values of pride, interpreting them as shame. Since player 1 cares about what his friends think of him, he is motivated by maximising their pride, which he estimates according to a pride dynamics function. While we do not go into details of estimation heuristics here, the intuition is that pride is estimated based on player 2's reactions, with *support* hinting at positive pride and other reactions suggesting a negative value (shame).

Utility functions of agents are as follows, where we use states where paths are expected under the assumption that states encode executions histories:

$$\begin{aligned}\mathtt{cu}^{\mathbf{s}}_{\mathrm{p_1}}(s) &= \lambda_b^{p_1}\mathrm{r}^{\mathbf{s}}_{\mathrm{b,p_1}}(s) + \lambda_t^{p_1}\mathrm{r}^{\mathbf{s}}_{\mathrm{t,p_1}}(s) + \lambda_p^{p_1}\mathbb{E}^s_{p_1}[pride_{p_2}],\\ \mathtt{cu}^{\mathbf{s}}_{\mathrm{p_2}}(s) &= \lambda_b^{p_2}\mathrm{r}^{\mathbf{s}}_{\mathrm{b,p_2}}(s) + \lambda_t^{p_2}\mathrm{r}^{\mathbf{s}}_{\mathrm{t,p_2}}(s) + \lambda_p^{p_2}[\![pride_{p_2}]\!]^s.\end{aligned}$$

Intuitively, player 1 is bold when $\lambda_b^{p_1} \gg \lambda_t^{p_1}$; in fact, we define their boldness as $\xi_{p_1} = \frac{2\lambda_b^{p_1}}{2-\lambda_p^{p_1}}$ and player 2's belief of player 1's boldness at state s is then $\mathbb{E}^s_{p_2}[\xi_{p_1}]$.

With that, player 2's pride is proportional to their belief of p_1's boldness in absolute terms, but when player 1 takes a timid move, pride is negative.

Experiments

For the first set of experiments, we assume player 1 ($\alpha_{p_1} = 30$) is more rational than player 2 ($\alpha_{p_2} = 10$), as we expect a group of people to make decisions more noisily than an individual. Additionally, we assume players do not discount future events ($\gamma = 1$) and are rather short-sighted ($\beta_{p_1} = 1$, $\beta_{p_2} = 2$). Moreover, as an attempt to recreate the setting from the psychological game, we fix goal coefficients of agents as follows: $\vec{\lambda}^{p_1} = [0.2, 0.35, 0.45]$, $\vec{\lambda}^{p_2} = [0.4, 0.2, 0.4]$. We then vary initial beliefs of agents and simulate the execution of the system for twenty steps. We find that, regardless of the accuracy of initial beliefs, within a few iterations the game reaches an equilibrium-like state characterised by dominance of *bold* actions followed by reactions of *support*. Interestingly though, this uniformity is disrupted by an occasional *suppress* reaction of player 2. We interpret it as complacency of player 2 who, having observed a streak of *bold* decision by their friend, assumes that this will continue in this manner whatever they do. In any case, our analysis suggests that, provided player 2 is given an opportunity to give feedback following their friend's decisions, out of the three psychological equilibria, the one characterised by $p = 1$ is superior.

Another experiment we perform shows that looking further into the future may produce better outcomes for everyone. In particular, we set goal coefficients of player 1 to $\vec{\lambda}^{p_1} = [0.35, 0.6, 0.05]$ so that he still prefers to be timid, but he now cares less about his friends' pride. We then simulate the execution for three different values of player 1's lookahead: $\beta_{p_1} = 1$, $\beta_{p_1} = 3$ and $\beta_{p_1} = 5$, and repeat this process five times to smooth out any probabilistic noise. Our findings, summarised in Table 6.2, show that by looking more into the future, player 1 can increase not only his, but also player 2's rewards gained along execution. Closer inspection reveals that longer lookahead allows player 1 to visualise growing pride of player 2, which outweighs player 1's natural preference for a timid move.

Overall, our treatment of bravery game serves as a comparison of our approach and psychological games framework. While both models stem from a desire to enrich the domain of utility functions, the way this is achieved differs. In our framework, emotions are captured by a complex mechanism that allows parameterisation and encoding of psychological theories, making it possible to capture intricacies of human feeling; in psychological games, emotions are reduced to beliefs about behaviour of others. Moreover, the CSMG framework is well-suited to capturing continuity

Table 6.2: Rewards accumulated by each player (p_1 and p_2) over twenty steps of bravery game, averaged over five runs, as a function of player 1's lookahead (β_{p_1})

β_{p_1}	p_1	p_2
1	4.63	2.11
3	5.39	6.22
5	5.43	6.71

of human preferences (via goal coefficients vectors), limits of our knowledge and the repeated, dynamic nature of our interactions. Conversely, the psychological games framework helps predict what behaviours are sustainable in the long-term, once beliefs of agents converge to reality.

6.2.3 Tipping

The next case study we analyse has to do with tipping (i.e., gratuity). In particular, our aim is to formalise the cognitive process that underlies a decision of how much to tip. This, in turn, allows us to infer various statistics, such as tipping norms in different countries, based on behavioural data.

Before presenting the details of the model, we make the following observations. First of all, tipping customs differ between cultures – what would be a very generous tip in Europe may qualify as an insult in America. We hypothesise that each country/region/community can be assigned a baseline amount, which characterises the social norm that dictates how much to tip having received a service of standard quality. Second, by tipping less than imposed by the local custom, an agent exposes themselves to feelings of guilt, which features as a mental attitude in our model. Finally, each agent's attachment to money differs and affects how much they're willing to tip.

Game Setup

Tipping is modelled as a two-person game between a service provider (we'll call them Ben) and service receiver (called Abi). Ben provides a service, be it waiting at a restaurant, a taxi ride or valeting, which we assume can be of *low*, *medium* or *high* quality. This is followed by Abi giving a tip, which ranges between 5% and 25% (we only consider multiples of 5 for simplicity). Figure 6.4 gives a graphical representation of one iteration of the tipping game; it is easy to deduce standard components of the model, such as the transition function T and set of actions

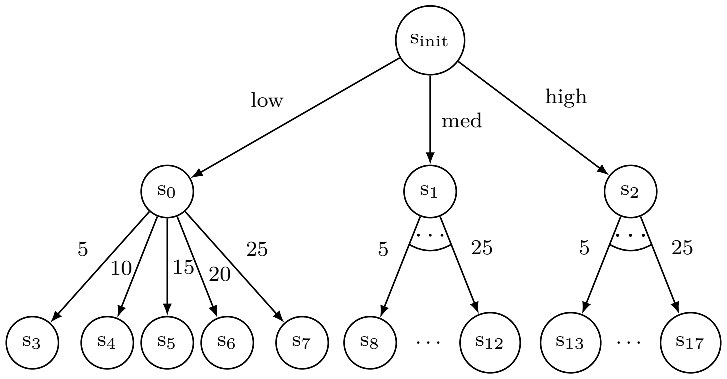


Figure 6.4: Tipping model

Act. As usual, states encode execution histories (which allows us to use states where paths are expected) so, for example:

$$s_1 = ([\mathit{med}], []),$$
$$s_6 = ([\mathit{low}], [20]).$$

Also, if Ben provides a service of high quality from state s_8 then the game would transition into state $([\mathit{med}, \mathit{high}], [5])$. A sole physical reward structure $\mathrm{R_m}$ models the monetary exchange associated with tipping with a state reward function $\mathrm{r}^{\mathbf{s}}_{\mathrm{m}}$ that assigns $-a$ to Abi and a to Ben in a state where Abi has just given a tip a. Note that an action reward would be more natural here, but mental rewards are collected at states and dealing with only one type of rewards simplifies analysis.

Mental Rewards As mentioned above, we hypothesise that guilt features as a factor when tipping. Note that, unlike the trust game, where an agent is driven by their opponent's trust, guilt is experienced by the agent themselves. Also, while in the trust game the behaviour of both agents is of interest, here we mostly concern ourselves with modelling Abi's behaviour; we assume Ben always does his best, but quality of the service provided varies. Therefore, our main task is to capture how much guilt, denoted η^g_{Abi}, Abi experiences depending on how high her tip is.

We assume that Abi's guilt is primarily influenced by two factors: (i) her guilt proneness, measured according to the GASP (Guilt And Shame Proneness) scale [117] as a number between 1 and 7, and (ii) her estimation of Ben's expectation of the tip, which we assume is a function of service quality. In particular, based on perceived quality of service received and a tipping norm, Abi estimates how high her tip should be. Tipping less than that reference value causes Abi to feel guilty ($[\![\eta^g_{\mathrm{Abi}}]\!] < 0$), and the smaller the tip, the more guilt she experiences (where the pace of guilt increase is exponential, but the rate is dependent on Abi's GASP

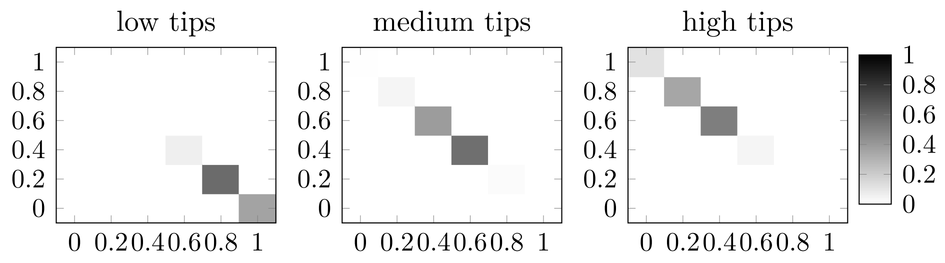


Figure 6.5: Posterior predictive distribution over Abi's goal coefficients for three different batches of data; λ_m^{Abi} on x-axis, λ_g^{Abi} on y-axis

score). Conversely, tipping higher than the reference value will invoke "negative guilt", i.e., pride ($[\![\eta^g_{\text{Abi}}]\!] > 0$), but the absolute values will be smaller than for guilt. With that, Abi's utility function captures her attachment to money and the guilt she would experience after tipping less than expected:

$$\mathsf{cu}^{\mathbf{s}}_{\text{Abi}}(s) = \lambda_m^{\text{Abi}} \mathrm{r}^{\mathbf{s}}_{\text{Abi}}(s) + \lambda_g^{\text{Abi}} [\![\eta^g_{\text{Abi}}]\!]^s.$$

Experiments

We now describe how using our tool with simulated tipping data can provide insights about population-wide patterns. The idea is that human actions are driven primarily by their personalities, beliefs and desires, but societal norms play a role too. For example, a successful home assistance robot must be able to adapt its behaviour not only to the preferences of its owner(s) but also to the environment it operates in. In the context of our example, tipping norm constitutes such a societal norm.

In all the experiments, we are only interested in learning Abi's parameters (and the tipping norm) – Ben's behaviour is assumed to vary probabilistically.

Inferring Parameters from Synthetic Data For the first experiment, we use three batches of synthetic data, each consisting of ten rounds of the tipping game. Tips in the first file are generally low, tips is the second file are mostly medium and tips in the third file are generally high, as judged by a human (more information about the synthetic data files is given in Appendix B). We perform inference on each of the data files, in which we learn Abi's goal coefficients, Abi's GASP score and the tipping norm. We take a (discretised) uniform distribution as a prior, separate for each parameter we are inferring (with around five elements in the support of each prior, see Appendix B for details). Note that computing disjoint posteriors is less accurate as it does not capture correlations between parameters, but it is easier to present and conceptualise.

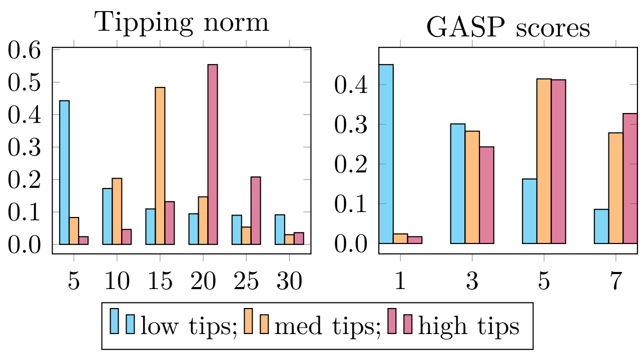


Figure 6.6: Posterior predictive distribution over Abi's GASP score and tipping norm

Figure 6.5 visualises inferences of Abi's goal coefficients; note that goal coefficients sum to 1, which explains why only points on one of the diagonals have positive probabilities. Reassuringly, the higher the tips, the more biased the posterior is towards λ_g^{Abi}. In other words, low tips are explained by an agent's attachment to money, while high tippers are inferred to place more importance on their feelings.

Next, Figure 6.6 depicts the visualisations of inferred distributions of the other two parameters, the tipping norm and the GASP scores. Clear patterns are observed, with the distributions of tipping norm peaking at growing values as the observed tips increase, as expected. The distinction between different GASP scores is not as clear cut, but suggests that low tippers are less prone to experience guilt, while those who give higher tips are almost certainly susceptible to begin culpable. However, GASP scores suggest no significant disparity between medium and high tippers, suggesting other factors, such as tipping norms, come into play.

We remark that it would be incorrect to infer the tipping norm solely based on the data (by taking an average of observed tips, say), tempting as it may seem, because parameters of an agent play an equally important role in determining their behaviour. To illustrate it, we use the data characterised by medium tips, and for a variety of combinations of Abi's goal coefficients and her GASP score (again, see Appendix B for details), we infer the tipping norm. To improve accuracy, we use a more informed prior on the tipping norm – still uniform, but its support is restricted to the set $\{5, 7, 9, 11, 15\}$.

The inferred posterior varies widely depending on the values of goal coefficients and the GASP score that we fix. For example, when Abi is more money-oriented ($\vec{\lambda}^{\text{Abi}} = [0.9, 0.1]$) and not prone to experience guilt (GASP score equal to 1), the posterior is biased toward smaller tipping norm (5 being most likely), but with little certainty. With a more centralised GASP score (equal to 4) and a lack of strong preference between guilt and money ($\vec{\lambda}^{\text{Abi}} = [0.5, 0.5]$), high tipping norm (15) is

Table 6.3: Prediction errors, aggregated over 10 runs; μ (resp. M) denotes the mean (resp. median); # is the number of rounds of tipping game used for learning

(a) Goal coeffs

#	MSPE	
	μ	M
0	.209	.174
5	.045	.035
10	.057	.041
15	.022	.015

(b) GASP score

#	MSPE	
	μ	M
0	10	10
5	5.38	4.91
10	7.37	5.61
15	4.01	2.25

(c) Tipping norm

#	MSPE	
	μ	M
0	107.5	112.5
5	105.33	102.82
10	69.18	68.55
15	53.06	37.39

deemed most likely and the confidence of the prediction is higher. In-between values of tipping norm also feature as likely for other values of Abi's parameters.

Inferring Parameters from Generated Data The other type of experiment we perform involves learning from data generated (through simulations) by the tool itself. For that, we randomly generate Abi's goal coefficients, her GASP score and the tipping norm and simulate the execution for ten rounds. We then attempt to learn the randomly chosen parameters starting from a uniform (discretised) prior, the same as before. We repeat this process ten times and evaluate each posterior predictive distribution by computing its mean squared prediction error (MSPE) (introduced in Section 3.2.5). We then aggregate the MSPEs from each run via mean and median averages.

Additionally, to study how the amount of data affects the accuracy of the inference, we repeat the above process on data generated by simulating five and fifteen rounds of the tipping game. The results are displayed in Table 6.3, separately for each inferred parameter. The leftmost column indicates the number of rounds that the data used for inference consisted of. For reference, the first row of each table shows prediction errors computed from the prior, before any learning occurs.

The results show that learning performs especially well for goal coefficients, even with relatively little data available. This suggests personality of an agent can be derived directly from their behaviour, with minimal confounding effect of other variables. On the other hand, tipping norm and the GASP score are harder to learn, reflecting the interplay of these two factors in determining agent behaviour. In particular, high tips may be explained by high GASP score even if the tipping norm is moderate, but they may also arise despite agent's low proneness to guilt when the tipping norm is high. This suggests considering a joint probability distribution of the two parameters, which we plan to analyse as part of future work.

6.3 Conclusion

This chapter has focused on describing the practical side of our framework, CSMG. We have outlined the main components of its probabilistic programming implementation and gave an insight into the representations we use internally. We have included key code fragments that capture the fundamental properties of our tool. Finally, we have presented a selection of case studies from a variety of domains, each highlighting a different aspect of our formalism. We have analysed each case study through a series of experiments, most of which involve randomly setting parameters of agents and running inherently stochastic simulations. This complicates the reproducibility of our experiments, as different resolutions of random choices may produce distinct results. While we have strived to counteract this by increasing the number of simulations (to smooth out irregularities) and providing a record of outcomes of random choices that occurred during our run, more effort is needed to make replicability more straightforward. In principle, setting random seeds should allow us to achieve that goal, but we have so far been unable to make it work as desired.

7 Human Experiment

Contents

In Chapter 5, we have presented the CSMG formalism, a probabilistic framework that models human-like decision-making processes. Then, in Chapter 6 we have described a software implementation of that model that supports behavioural predictions and inference of preferences. We claim robots can be endowed with this formalism to help them make sense of the world and advance their partnerships with humans. This chapter substantiates our assertion by reporting results of a human experiment we have run to evaluate our framework.

We begin by describing the design of our experiment and its objectives in Section 7.1. Our method of data collection involves a custom, highly interactive web application – we discuss its architecture and selected implementation details in Section 7.2. Finally, Section 7.3 presents the results of the study and the conclusions we have drawn from them are outlined in Section 7.4.

7.1 Experiment Design

In whatever setting our human study would be run, its main objective was to evaluate the predictive power of our framework. However, depending on the environment of the experiment, our preference was to study aspects of human-robot interaction that could inform future robot designs. Therefore, objectives of the study go hand in hand with the experimental setup we adopt. Below we describe the choices we have made and motivate them.

7.1.1 Introduction

To enable evaluation of our model, we needed a scenario where humans make decisions (which we would try to predict using our framework), ideally as part of direct interaction with a robot (which we could implement using our software). We wanted this scenario to be easy to understand for participants and fairly simple to implement (due to ongoing pandemic and time constraints, conducting the experiment online was the only viable approach, necessitating development of a custom web application). Fortunately, there is a two-player game that readers of this thesis will be familiar with that fit that bill perfectly.

Indeed, the trust game is widely accepted as a standard experimental setup for measuring trust and trustworthiness through actions, rather than self-reports. It is simple enough for a human to grasp within minutes, but features a powerful, monetary incentive mechanism. Its appropriateness is attested by its popularity as an experimental paradigm, with well over a hundred human studies based on trust game performed to date (see Johnson et al [54] for a meta-analysis).

On the other hand, measuring trust and trustworthiness through trust game has its critics too [118]. It is often pointed out that a decision of how much to invest (resp. return) cannot be attributed solely to trust (resp. trustworthiness). Indeed, by running cleverly designed variations of trust game, it has been suggested that other social preferences, such as altruism, inequity aversion or betrayal aversion, influence participants' decisions [119, 120]. Moreover, lack of stability of the paradigm is raised as a concern; in particular, the following factors of the trust game vary widely between studies: (i) the investment multiplier (K), (ii) the horizon of the game, (iii) the way actions are elicited, and (iv) framing of game description and instructions.

While we agree with all of the above, we believe it does not disqualify trust game as an experimental paradigm. However, the critique provides important insights that inform the way in which we employ the trust game. In particular, at no point do we draw direct correspondence between investment decisions and trust. Instead, we encode our hypothesis of how trust affects decisions in our model and test that hypothesis by running our experiment. We also carefully consider what setting of factors of the game is most appropriate for our purposes (detailed in Sections 7.1.3 and 7.1.5).

Moreover, despite the high number of replications of the trust game experiment, we have identified a gap in the literature. Namely, a vast majority of experiments conducted to date involve humans interacting with other humans. Only recently some authors set out to study if, and how, human behaviour in the trust game would be different if they were to face a robot. We have identified three such studies and described them in more detail in Section 2.3. The overarching conclusion that can be drawn from these investigations is that humans trust robots at least as much as other humans.

However, a major limitation of the aforementioned studies is a lack of autonomy of the machines and the one-offness of the interaction. Therefore, we set out to fill that gap by endowing the robot in our experiment with a sophisticated decision-making mechanism based on our framework. Moreover, to study the dynamics of trust in human-robot interaction, we have participants play repeated trust game. Also, unlike all past studies which put the human subject in the role of an investor, we select the role at random for each subject. We defer more details about our experimental procedure until Section 7.1.5.

Note that, throughout this chapter, we refer to the artificial agent we designed to play the trust game against human participants as a *bot*. This reflects the non-physical form of our agent, which was the only practical way for us to proceed. Moreover, we drop the naming of players we have used until now, Alice and Bob, and instead refer to players as investor and investee. We also remark that, since the experiment involved humans, ethical approval was sought and granted by the Ethics Committee of the Department of Computer Science at University of Oxford (ref. CS_C1A_21_029).

7.1.2 Data Collection

One of the first things we had to settle on was the platform on which to run our experiment. Given the ongoing pandemic, Amazon Mechanical Turk (below abbreviated as MTurk) was an obvious choice. It is a crowdsourcing marketplace that allows *requesters* to submit so-called Human Intelligence Tasks (HITs), which can subsequently be completed by *workers* in exchange for pay. Upon a worker completing a given HIT, the requester gets to approve or reject that assignment. The most common tasks to be found on the platform are short and simple; think of identifying objects in images, transcribing audio or validating spreadsheet entries. However, the platform has increasingly been taken up by researchers to conduct scientific experiments with human subjects, a trend accelerated by the COVID-19 pandemic. Using the MTurk platform comes with several benefits:

- **Diversity**: respondents' demographics is more diverse than the typical college campus population
- **Scalability**: arbitrarily high number of participants can be requested with no extra time or resource expenditure
- **Economy**: both in terms of time and money
- **Automation**: an API is provided which allows automating all the admin tasks, such as approving assignments or paying subjects

Of course, Mechanical Turk has its disadvantages too. Of these, two are most prominent. Firstly, any task submitted to the platform must be possible to complete using only a computer with access to internet. This is problematic for many medical studies that require brain scanning or use of other specialised machinery. It would be problematic for us if we wanted to use real robots. While some may argue that doing that would make our results more credible, it would also come with many difficulties. Moreover, autonomous machines will come in various forms and shapes, meaning that experimental results obtained by using a humanoid robot may not necessarily transfer to driverless cars. Therefore, by using a non-physical bot in our experiment we focus our attention on humans' relationship with autonomous automation as a whole, be it a robot vacuum cleaner or a military drone (perhaps to a lesser extent).

Secondly, a major disadvantage of MTurk is the design of the incentive system of the platform. The majority of workers are there for the money and are highly skilled at completing the tasks in the shortest amount of time possible. This can negatively affect the quality of the data collected, especially if no steps are taken to filter out inattentive workers. There are two standard practices to ensure data quality. First of those is provided by the platform itself, which allows one to target the study only at workers of highest calibre. In particular, a subset of MTurk workers is awarded a so-called Masters Worker qualification. Quoting MTurk webpage, Master Workers "have consistently demonstrated a high degree of success in performing a wide range of HITs across a large number of Requesters". It is also possible to filter workers using other metrics, such as HIT approval rate, which measures the percentage of tasks completed by a worker that were approved by the requester. It is customary to place both of the above requirements on workers participating in research studies and this is what we do – we required workers to be Master Workers and have their approval rates above 95%.

Another common technique applied to ensure high quality of submissions involves including attention or comprehension checks as part of the task. We implement two such mechanisms as part of our experiment design: (i) a time control for behavioural questions that we ask subjects before they play the game and (ii) a set of comprehension questions that check participants' understanding of the trust game. We describe those measures in more detail in Section 7.1.5.

MTurk comes with a wide range of predefined templates for surveys, image/video classification tasks, data collection, etc. However, for a very specific application such as ours, a custom solution was required. A typical approach in such cases is to use the MTurk platform in a minimal way. Namely, a link to an external site on which the experiment is hosted is included in task description. Upon completing the work on the site, a participant is provided with a unique code which they then input on the MTurk task page before submitting the task. More detail on how we use Mechanical Turk is given in Section 7.2.

Before moving on to the objectives of the experiment, we explain some nomenclature we use throughout this chapter. Every instance of trust game played between a subject and our bot leads to an outcome that is a sequence of investments and returns that occurred in the game. We introduce two properties of outcomes that will be useful in evaluating and comparing plays between participants. Firstly, *total welfare* of a game is the sum of earnings of both players; it is proportional to the size of investments. Second, *fairness* of an outcome is an appropriately scaled ratio of earnings of both players; in particular, supposing investor earned $\$k$ and investee earned $\$l$, fairness is given by $\phi = \frac{k-l}{k+l}$. Hence, its value ranges between -1 and 1, with positive values corresponding to outcomes where investor earns more than investee, and vice versa.

7.1.3 Objectives

Having given an overview of our experimental setup, we now state the objectives of our study.

Model Validation As stated above, our most important goal was to evaluate the predictive power of our framework. For each play, at every step, the bot generates predictions of the participant's action as part of computing its own action. Those predictions are probabilistic, meaning that they take the form of a probability distribution over possible actions. We compare the predictions generated by our model (identified as CSMG) to the following baseline predictors:

- UNIFORM: predicts each action is equally likely
- LAST: predicts next action will be the same as the last action
- AVERAGE: predicts next action will be the average of all previous actions
- DANG: a predictor based on computational trust model by Dang et al. [121]

The first three predictors are very simple, but DANG deserves an overview. The main idea behind it is to capture how well a user behaved in the past with a single measure, called the trust score, which is then used to predict future behaviour. The trust score is updated in every round of the game using an intricate system of equations manually crafted to encode various hypothesised properties of trust. This includes common assumptions about the dynamics of trust that can be captured by a logarithmic function (which is smooth, increasing and concave on $[0, 1]$ and crosses identity at the extremes). Moreover, authors make an effort to ensure their measure punishes so-called fluctuating behaviour, characterised by most recent sending proportion significantly deviating from those in the past.

With that, the trust score is based on the weighted average of (i) the current trust, which is roughly proportional to the most recent sending proportion (defined as the ratio of actual transfer and maximal possible transfer):

$$current\_trust_t = \log(send\_proportion_t \times (e-1) + 1),$$

and (ii) aggregate trust, which captures all past behaviour and is updated after every round, with a stipulation that the most recent transfer matters more the more it deviates from previous transfers:

$$aggregate\_trust_t = \alpha_t \times current\_trust_t + (1-\alpha_t) \times aggregate\_trust_{t-1},$$

where $\alpha_t \propto |\mathit{current\_trust}_t - \mathit{current\_trust}_{t-1}|$. Then, the weighted average is expressed as

$$\begin{aligned}\mathit{expect\_trust}_t = \mathit{trend\_factor}_t \times \mathit{current\_trust}_t \\ + (1 - \mathit{trend\_factor}_t) \times \mathit{aggregate\_trust}_t,\end{aligned}$$

where $\mathit{trend\_factor}_t$ represents the recent trend in user's behaviour; higher values mean sending proportions have improved and vice versa. Finally, the trust score is expressed in terms of $\mathit{expect\_trust}_t$ as follows:

$$\mathit{trust\_score}_t = \mathit{expect\_trust}_t \times \mathit{change\_rate}_t,$$

where $\mathit{change\_rate}_t$ measures fluctuating behaviour of the user; it is close to 1 when transfers are consistent or deviate slightly, but quickly approaches 0 if divergence persists. Moreover, $\mathit{change\_rate}_t$ decreases faster as a response to sudden decrease in sending proportion than when transfers increase, which is still punished, however.

In summary, DANG predicts future behaviour of a user based on a trust score that measures past behaviour of that user and whose formulation detects fluctuating behaviour. DANG shares many features with the trust dynamics model presented in Section 5.4.4 on which our predictor, CSMG, is based. However, there are two major differences: (i) DANG does not take into account opponent's behaviour and (ii) DANG does not differentiate between the two roles in the game.

Note that UNIFORM is a probabilistic predictor, same as CSMG, while others are deterministic. Now, a natural measure to evaluate a probabilistic prediction is mean squared prediction error (MSPE), introduced in Section 6.2.3. It is easy to see that computing MSPE of a deterministic prediction (treated as a special case of a probabilistic prediction) is equivalent to computing its MSE.

However, there is an important discrepancy between stochastic and deterministic predictors. The former are represented by discrete probability distributions whose support is the set of available actions. On the other hand, DANG and AVERAGE formulate predictions as fractional numbers, which are incompatible with the expected format of a probabilistic prediction and cannot be evaluated using MSPE.

Therefore, to find common ground for evaluation, we reduce each probabilistic predictor to a deterministic (fractional) one by taking an expectation of the probability distribution that constitutes the prediction. Hence, every prediction is reduced to a single, possibly fractional number and evaluated by computing its mean squared error.

Determining Optimal Robot Parameters Recall that every agent in our model is characterised by a vector of goal coefficients. As a result, every time the model is implemented in a robot, parameters' values must be set appropriately. The second goal of our experiment is to study how various settings of the bot's goal coefficients influence game outcomes and satisfaction of participants. Our hope is that our findings lay a foundation for future robot designs.

More specifically, we define three types of bots, each characterised by a unique combination of goal coefficients:

- a greedy bot, with goal coefficients vector $\langle 0.8, 0.2\rangle$
- a neutral bot, with goal coefficients vector $\langle 0.5, 0.5\rangle$
- a selfless bot, with goal coefficients vector $\langle 0.3, 0.7\rangle$

For each participant, the type of the bot is selected at random. We then study how the total welfare and fairness of plays differ between the three cohorts. We also query subjects' impressions of the bot and compare between treatments.

Prior Effect The third objective of our experiment is partially inspired by Wagner et al. [40], who propose a conceptual framework for human-robot trust but find it fails to detect overtrust. We hypothesise a possible solution to the problem could be to include priors about individuals. To test this hypothesis in the setting of our framework, we consider two treatments which we refer to as UNIFORM PRIOR and INFORMED PRIOR. To understand the difference between the two treatments, recall that an agent in our framework, besides their own parameters, is also equipped with an initial state. It consists of a belief about every other agent, as well as estimations of opponents' mental states. In other words, an initial state of an agent is made up of priors about all other agents. Now, when an agent knows nothing about one of their opponents, a uniform prior is in order. However, any relevant information that is available may be encoded in agent's initial state, resulting in a more informed prior.

In the context of the trust game, bot's prior over their human opponent consists of a belief that essentially captures bot's trust towards that opponent, bot's initial estimation of human's trust and bot's estimation of human's meta-parameters (rationality, lookahead and discount factor). To generate such a prior, we include as part of our experiment a questionnaire that subjects fill in before playing the game. Then, for each participant, we randomly decide whether to use their answers to the questionnaire to generate an informed prior or use a uniform prior. This produces UNIFORM PRIOR and INFORMED PRIOR treatments. Details about what questions make up the questionnaire and how we use the answers to generate a prior are in Section 7.1.5.

Human Opportunism An important decision that has to be made as part of a repeated trust game experiment is whether to disclose the horizon of the game to the participants. It is often hypothesised that revealing the horizon will lead the human to alter their behaviour in the latter stages of the game [121]. We decided to test this assumption by including two treatments, one where the horizon is disclosed (DISCLOSED) to the participant and the other where it is not (UNDISCLOSED). To evaluate it, we restrict our attention to games where the subject plays as investee and we measure how their last action differs between the two treatments. We hypothesise that the last return will be on average smaller (when considered as proportion of investment) in the cohort with the disclosed horizon, showing opportunistic nature of humans when dealing with robots.

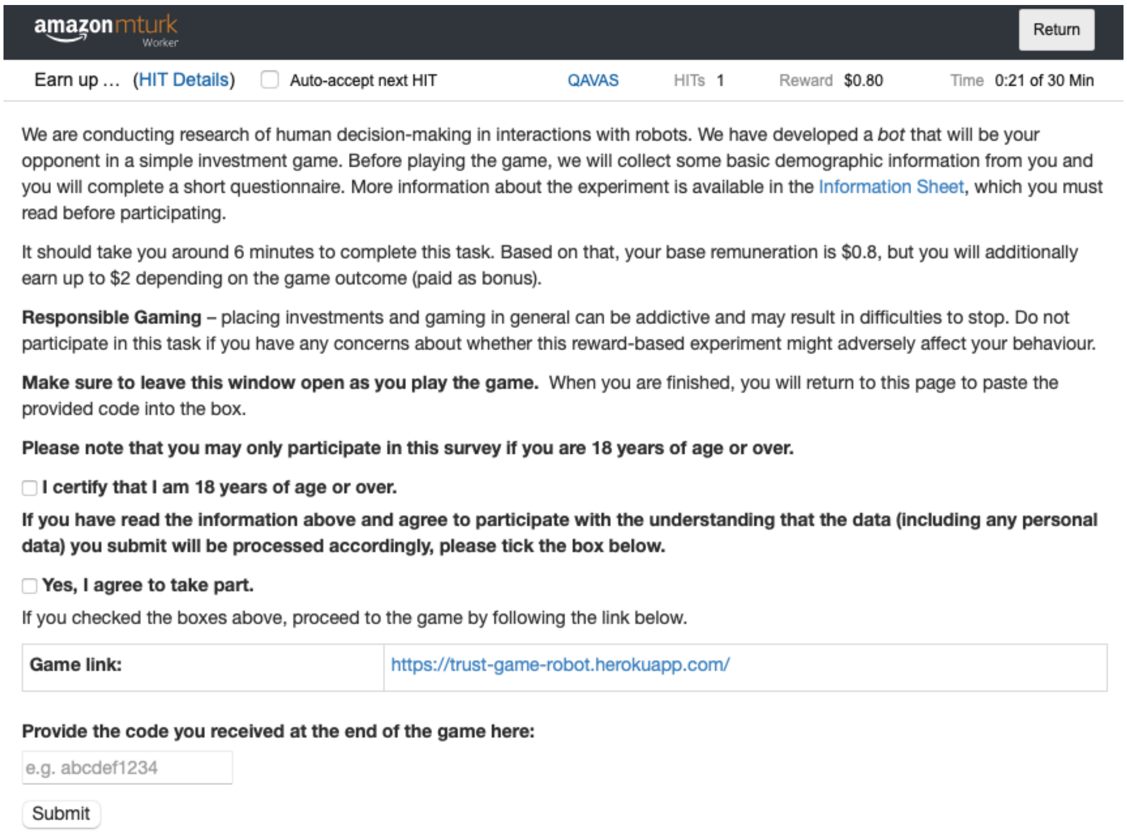
7.1.4 Novel Contribution

According to the best of our knowledge, the experiment proposed below is the first to date where humans play repeated trust game against a machine that uses a sophisticated reasoning mechanism. The three aspects which make our approach unique are (i) the game is repeated, i.e., played over multiple rounds, (ii) the robot (or, more precisely, *bot*) that participants play against is driven by our probabilistic, cognitive framework, and (iii) we will run the bot not only as an investee (which is what all the experiments mentioned above do), but also as an investor.

7.1.5 Experimental Procedure

Having given an overview of our experimental setup and a list of objectives, we now proceed to describe in detail our experimental procedure. This section covers the high-level aspects, while implementation details are to be found in Section 7.2.

As mentioned above, participants for our experiment are recruited using Amazon Mechanical Turk marketplace. To qualify for this assignment, a worker on MTurk platform must be in possession of the Master Worker qualification and have HIT approval rate above 95%. Eligible workers are presented with a task page displayed in Figure 7.1. It contains basic information about the experiment, a few disclaimers, informed consent form and, most importantly, a link to a web application where the experiment is hosted. That is where all the interaction with the participant takes place and we describe it in detail in what follows.



We are conducting research of human decision-making in interactions with robots. We have developed a *bot* that will be your opponent in a simple investment game. Before playing the game, we will collect some basic demographic information from you and you will complete a short questionnaire. More information about the experiment is available in the Information Sheet, which you must read before participating.

It should take you around 6 minutes to complete this task. Based on that, your base remuneration is $0.8, but you will additionally earn up to $2 depending on the game outcome (paid as bonus).

Responsible Gaming – placing investments and gaming in general can be addictive and may result in difficulties to stop. Do not participate in this task if you have any concerns about whether this reward-based experiment might adversely affect your behaviour.

Make sure to leave this window open as you play the game. When you are finished, you will return to this page to paste the provided code into the box.

Please note that you may only participate in this survey if you are 18 years of age or over.

☐ **I certify that I am 18 years of age or over.**

If you have read the information above and agree to participate with the understanding that the data (including any personal data) you submit will be processed accordingly, please tick the box below.

☐ **Yes, I agree to take part.**

If you checked the boxes above, proceed to the game by following the link below.

Game link:	https://trust-game-robot.herokuapp.com/

Provide the code you received at the end of the game here:

e.g. abcdef1234

Submit

Figure 7.1: Experiment task page on MTurk

Web Application

The web application, which was developed specifically for this experiment, can be conceptually divided into five parts, reflecting the flow that a user follows when completing our study. The first component is a set of standard demographic questions that give us an idea of what population our respondents belong to. Upon answering those, the user is presented with a study-specific pre-game questionnaire designed to estimate relevant aspects of participant's decision-making process and their personality. Having completed the questionnaire, the subject is shown instructions explaining the game and asked questions that check their understanding of game rules. Participants are given two chances to answer those questions correctly; in case they fail, they are unable to proceed to the game and their participation in the experiment ends. However, to reimburse them for the time they had already committed, subjects are paid between $0.20 and $1, depending on how long they spent trying to understand the game.

In case they answer correctly, each participant then proceeds to the game, in the role of an investor or investee, whichever was randomly selected for them. The play always lasts seven rounds, though only subjects in the DISCLOSED treatment are aware of that. The number itself was chosen to strike the right balance of keeping participants engaged during the whole play while ensuring enough data is obtained to provide insights into trust dynamics and human nature. Moreover, an odd number was deemed less likely to be correctly guessed by subjects in the UNDISCLOSED treatment. Further, we mention that the multiplication factor K of the game is set to 2. This largely stems from personal preference – we believe it creates a more straightforward dynamics, with an expectation on the investee to return half of what they receive. In any case, we are of opinion that the value of the multiplication factor, be it 2 or 3, is of little significance.

Finally, once the game is complete, we ask participants three simple questions about their experience and give them an opportunity to provide feedback. Upon submitting their answers, a unique code is generated for each user which, when entered on MTurk task page, will ensure they get paid appropriately. We now describe each component of the web application in detail.

Demographic Questions We ask subjects to provide their age, gender, nationality, education and previous robot exposure as part of a standard demographic survey.

Pre-Game Questionnaire The questionnaire that follows the demographics survey consists of six questions that attempt to determine each participant's lookahead, rationality, trust and trustworthiness.

Time Control Note that the questions, especially the first four, are cognitively challenging and designed to induce subjects to think hard. To ensure that participants read the questions fully and carefully consider their answers, we implement a time control mechanism. Based on its length and difficulty, each question is assigned an expected time (ranging from 10 to 25 seconds) needed to answer it. A subsequent question is displayed only once that time elapses. Participants are informed how much time they have for each question and how long remains until the next question appears. We evaluate whether the time control is effective in improving quality of answers in Section 7.3.4.

Lookahead The first question is inspired by the 11-20 money request game [122], originally devised to measure depth of reasoning that an individual performs:

> Imagine a following game: There are two players and each requests an integer amount of money between \$11 and \$20. Each player will receive the amount they requested but a player will receive an additional amount of \$20 if they request exactly one dollar less than their opponent. What amount would you select in such a game?

The idea is that, the lower the number an individual chooses, the deeper their reasoning. For instance, selecting $18 may be a result of a reasoning process along the following lines: "My opponent will expect me to select $20; hence they will select $19; therefore I should choose $18 to obtain the bonus". Using nomenclature of our framework, each answer is associated with a lookahead value (e.g., agent that looks three steps ahead would compute $17 as optimal choice).

The way we estimate an agent's lookahead based on their answer is given in detail in Appendix C, but the idea is the following: we use the base distribution reported by Arad et al. [122] for the UNIFORM treatment, with a stipulation that the support of the distribution must be adapted to ensure reasonable response times of our app. In the INFORMED treatment, the estimation is obtained by soft conditioning (using WebPPL's `factor` operator) base distribution on a lookahead value that corresponds to the observed answer.

Rationality To estimate participant's rationality, we present them with three sets of lotteries of increasing complexity and query their preferences. This is the most cognitively challenging part of the questionnaire and spans three questions (one per lottery set). Full details are again given in Appendix C; in the nutshell, each question solicits preferences between two lotteries, each specifying various probabilities of receiving various amounts of money (all lower than $100). The assumption is that a perfectly rational agent would correctly compute expected dollar value of each lottery and rank them accordingly.

The way a rationality estimation is computed based on participant's answers is as follows: we start with a base rationality value of sixteen and we increase (resp. decrease) it for every correct (incorrect) answer, where the magnitude of the modification is inversely proportional to the difficulty of the question (i.e., easier lottery questions matter more). The resulting rationality value is between zero and forty-four. See Appendix C for a formal algorithm.

Trust Participant's trust towards our bot is estimated by asking the following question:

On a scale from 1 to 5, how would you describe your overall trust towards robots?

Since the participant has no additional knowledge about our bot, we assume that their trust towards it is the same as their overall trust towards robots in general. Recalling that trust (in particular core trust, which is of relevance here) in our framework is a number between zero and one suggests a straightforward, linear mapping between an answer to the above question and a trust value. Recall also that agent's estimation of their opponent's mental state is represented in our framework as a probability distribution over possible values of that mental state. In this case, we use a Dirac distribution centred on the trust value given by the aforementioned mapping.

Trustworthiness (Initial Belief) The last question in the pre-game questionnaire has been designed to assess participant's trustworthiness, or equivalently, their goal coefficients. The wording of the question is not as direct as for trust:

On a scale from 1 to 5, how concerned are you with the way the robot perceives you?

Our choice of phrasing reflects the fact that, in our framework, trustworthiness of agent A towards B measures how much A cares about B's trust. Trustworthiness, just like trust, is measured on a $0 - 1$ scale. Recall that, in our model of trust game, trustworthiness is given by the ratio of agent's goal coefficients. Estimation of participant's goal coefficients is captured by a belief which takes the form of a Dirichlet distribution, represented as a vector of positive real numbers $\vec{\alpha}$.

Since the ratio of those real numbers is equal to the expected ratio of goal coefficients, i.e., the trustworthiness, subject's answer to the above question should be correlated with the ratio of Dirichlet parameters. It then remains to choose their sum, which is inversely proportional to the uncertainty of our belief. After some experimentation, we settled on setting the sum to 3. This results in initial beliefs such as $[2, 1]$ (when the answer is 1), $[1.75, 1.25]$ (when the answer is 2) or $[1.25, 1.75]$ (when the answer is 4). In the UNIFORM condition, belief is set to $[1, 1]$, reflecting greater uncertainty about participant's trustworthiness. Appendix C describes in more detail how belief is set based on subject's answers and why it is done this way.

The Game Upon completing the pre-game questionnaire, the subject proceeds to the game itself. It is during this transition that the participant is randomly assigned to one of the treatments; namely, the character of the bot is selected (SELFLESS, NEUTRAL or GREEDY), along with the prior (INFORMED or UNIFORM) and horizon (DISCLOSED or UNDISCLOSED) and the role of the participant. It is also at this point that the state of the bot is initialised, either based on questionnaire answers (in the INFORMED condition) or otherwise (UNIFORM condition).

Game Description The game screen initially consists of the game description and a set of three comprehension questions. The explanation of the game has been designed with brevity and clarity in mind. It has been an iterative process, heavily influenced by the feedback gathered from volunteers who tested the game. Note that the description differs depending on the role of the participant.

Comprehension questions have been chosen to cover as many aspects of the game as possible without being overly complicated. subjects are given two opportunities to answer the questions correctly. This is to avoid punishing participants in case they accidentally choose a wrong answer (as a result of misclicking, misunderstanding the question or otherwise) on their first attempt. Full text of the game description along with comprehension questions can be consulted in Appendix C.

Game Play If the participant answers the comprehension questions correctly (on the first or second attempt), they proceed to the game. It always lasts seven rounds, but the subject is only told that number in the DISCLOSED condition. The game interface is designed to be simple and error-free, hence selection of user's action is queried with buttons rather than text fields. Also, given the potentially long-running computation of bot's action, a spinner is activated for its duration to keep users informed.

Post-Game Questionnaire Once play is complete, a summary of the game is presented to the participant; it consists of their earnings in the game and their payment (see Section 7.1.5). Moreover, we ask three simple questions about their experience. The first question queries subject's assessment of how human the bot's behaviour felt:

To what extent do you agree with the following statement:
Playing with the bot felt like playing with another human.

Possible answers include complete or partial (dis)agreement or indifference.

The other two questions relate to participant's trust changes as a result of playing the game. One concerns subject's trust towards the bot they played the game with:

Has your trust towards this particular bot changed after playing the game?

while the other one checks whether the trust change (if any) translates to all other robots:

Has your general perception of robots changed?

Apart from answering the questions, participants are given a chance to provide some feedback. Upon submission, they receive a unique code which they then have to enter on the MTurk task page.

Payment Structure

We now discuss the remuneration subjects receive for taking part in our study. As mentioned in the introduction, our goal from the onset was to have a variable remuneration structure so that participants are motivated to earn more in the game. However, ethical considerations require us to pay subjects even if they do not earn anything in the game.

As a result, remuneration is split into base payment, which every participant receives regardless of game outcome, and a bonus, which reflects each subject's earnings in the game.

The base payment has been chosen so as to ensure that subjects are paid consistently with the UK National Minimum Wage, in compliance with ethical approval. In particular, after converting to dollars (only available currency on MTurk), the base payment had to be equivalent to \$12 per hour. Based on trial runs, we determined that completion time rarely exceeds ten minutes and averages below that and set the base payment to \$2.

Determining the range of bonuses revolved around a trade-off between keeping the cost of the study reasonable while trying to strongly motivate subjects to take the game seriously. In the end, we settled on an expected bonus value of a little over \$1, so that it makes up more than a third of the participant's expected income.

In particular, the currency in the game is introduced as abstract "units", rather than dollars. At the start of each round, the investor is endowed with 4 units and whatever they invest gets doubled. Hence, the maximal possible earnings of each player during the seven rounds of trust game are 56 units. This, of course, is highly unlikely to occur in practice. Indeed, players' earnings rarely exceed 28 units, as typically not all investments are maximal (hence there is less than 56 units in total) and earnings of the two players do not usually differ significantly. Units are converted into dollars using the exchange rate of 1 unit = \$0.05, which participants are made aware of prior to play.

In fact, the structure of the base payment is a little more complicated than stated above. This reflects the possibility of a participant failing the comprehension test and being unable to proceed to the game. We deemed it unethical to leave such a subject with nothing, even though they do not provide any useful data to us. However, we also want to differentiate between subjects who make a true effort to understand rules of the game but are unable to, and subjects who rush completing comprehension check in hope of getting lucky.

We achieve this as follows: we set the guaranteed base payment that every subject receives to \$0.2. Now, if a participant passes the comprehension check, their base pay increases by \$1.8 to \$2 (reflecting the time they are expected to spend on the experiment). On the other hand, if they fail, their payment depends on the time it took them to try to understand the game (in particular, the time elapsed between being presented with game instructions and submitting their answers to comprehension questions for the second time). Specifically, the additional payment (on top of the \$0.2 base) is nothing if time spent comprehending is less than one minute and increases linearly by \$0.2 for every additional minute spent, up to four minutes. For example, if 2:24 min was spent, extra \$0.4 is paid and if 5:30 min was spent, extra \$0.8 is paid, which is the maximal amount. In summary, participants who fail the comprehension check are paid between \$0.2 and \$1, depending on how much effort they put into trying to find the correct answer.

7.2 System Architecture & Implementation

Previous sections described in detail the high-level design of our experiment. However, implementing the required mechanisms was far from straightforward. This section overviews the architecture of the web application we have developed to conduct this study and the challenges we had to overcome.

7.2.1 Web Application

We have developed a web application that implements the experimental procedure described in Section 7.1.5. A typical web application consists of a backend (application code that runs on the server), frontend (application code that runs on the client, i.e., the browser of a user visiting the app) and a database and must be hosted on a reliable platform to ensure reliability. Below we overview the technological choices we made for each of these components.

Backend Back in the early days of web development, the number of technologies available was quite limited and so the choice was rather straightforward. Many web applications were built using PHP for backend, HTML and Javascript for frontend and an SQL database for permanent data storage.

However, times have changed and today a plethora of web frameworks written in various programming languages exist. Fortunately, our choice was made easier by the fact that our model is implemented in WebPPL, which in turn is written in Javascript. Any WebPPL program can be executed from within Javascript code. Now, running our model involves heavy computation which may take many seconds to complete. Such jobs should be run on the server side to ensure reliable performance and good user experience. This meant that the backend of our application had to be implemented in Javascript.

Fortunately, Node.js is a popular and widely supported Javascript runtime environment which is highly suitable for backend development. Many frameworks exist that facilitate building web applications. One of the most popular ones, thanks to its flexibility and minimality, is Express.js. It makes the job of a programmer much easier by taking care of common tasks such as routing HTTP requests and responses and error handling. That enables the developer to solely focus on defining an appropriate API and implementing its functionality. We have chosen Express based on recommendations on the web.

Frontend Similarly as for backend, we have also used a web framework to aid development of client-side code. Frontend frameworks are not as essential as backend ones since writing HTML (which is what frontend is about) is much less complex than connecting to sockets and parsing HTTP requests (which is what the server must do). Still, they have gained a lot of popularity recently as they make apps faster, more scalable and easier to develop.

One such framework is called React and, besides improving performance, it brings features from object-oriented and functional programming into frontend development. React forces the programmer to think of the user interface as a hierarchy of stand-alone, reusable components, each with its own state and functionality. This makes frontend code much more readable and easier to debug. While we had not known all this when we were choosing which framework to use, we selected React due to its popularity and for personal development.

Database That leaves us with the last major component of a modern web application – the database. All the interactions that participants of our experiment have with our app, particularly their choices in the trust game, must be securely stored for subsequent analysis. We have decided to use for this purpose Cloud Firestore, which is part of a wider platform called Firebase, developed by Google to aid development of web and mobile applications. Firestore is their second-generation NoSQL database in which data is stored as part of documents, which in turn belong to collections. Each document has a JSON format, supporting a variety data types such as numbers, strings, arrays or timestamps. Firestore is free to use as long as usage does not exceed certain, quite generous limits. It offers an intuitive web interface for viewing and managing the data, as well as APIs to access the database programatically.

Hosting Once all the components of a web application are chosen, it must be decided where the app is to be hosted. Heroku is a well-established platform which, crucially, supports Node.js applications and makes it very easy to manage, configure and deploy them. Moreover, it offers flexibility with respect to utilisation of computational resources, bringing significant cost savings. This is especially relevant in our case, given that our application is only expected to receive high traffic for the duration of the experiment, which was expected to last no more than a few days. At all other times, we can utilise the free tier, enabling us to develop and test the application in production environment without incurring any costs.

7.2.2 Mechanical Turk

In this section, we give a technical overview of how we use the Amazon Mechanical Turk platform. Recall that, once a HIT (Human Intelligence Task) is released on MTurk by a requester, workers can view it and have a chance to accept it. If they do, they have a limited amount of time (set by the requester) to complete and submit the task. Typically, tasks are short and can be performed without leaving the MTurk platform. However, unconventional HITs such as ours require a custom solution which workers carry out on a separate site (such as our web application). Once they are finished, workers submit the task, which must subsequently be approved or rejected by the requester within a limited time, declared by the requester prior to publishing the task. In case the requester does not make a decision within allotted time, the HIT is automatically approved and the worker paid the assignment fee. If the requester approves the assignment, they may pay the worker a bonus of any size. If the requester rejects the assignment, worker is not paid.

In our case, we would like to approve an assignment provided we can verify that the worker has completed the experiment. Recall that, since we want to reward participants who fail the comprehension test, they are also classified as completing the experiment, along with those who complete the game play. To make this verification possible, every time a subject completes the experiment, we present them with a unique code, consisting of ten random alphanumeric characters. Internally, this code is created as soon as a user completes the pre-game questionnaire and serves as an identifier of that user throughout the game. This means that all requests originating from that user's browser are accompanied by their identifier and all their interactions (questionnaire answers, actions in the game) are saved in the database in a document identified by that code. The worker is asked to enter the code in a text box inside the task page on MTurk so that, once they submit the HIT, we can retrieve the code they had entered and perform the verification using information saved in the database.

This process may sound straightforward, but manually performing this verification for more than hundred experiment participants would be highly time consuming. Fortunately, MTurk provides an API which allows one to perform all the necessary operations automatically, from creating and publishing the HIT to approving and rejecting assignments.

HIT Creation When our server (re)starts, several things happen as part of initialisation. We check that the connection to the database is live; we set up logging according to a value of an environment variable; and, if enabled by another environment variable, we use MTurk API to create a HIT (or retrieve an existing one). This operation, if it succeeds, returns an identifier of the created HIT. This long string of characters serves as the name of a database collection that will store all the data gathered during that run of the application (interaction of each user is stored in a separate document).

Creating a HIT, which is an operation offered as part of MTurk API, involves specifying its name and short description as well as the HTML code that will constitute the task page. The task page contains a slightly longer description of the task, a link to an information sheet with more details about our study, a few disclaimers, informed consent form, link to our web application and a text box where the code provided upon completion is to be entered. Additionally, various configuration items are set as part of task creation, such as workers' qualification requirements, how long the task should be live on the platform or maximum number of completed assignments.

Once the task is live, every five minutes, our server checks if any new assignments have been submitted by calling an appropriate API endpoint. In case some are found, they are processed in turn. We start by retrieving the code that the worker entered. We then check whether there exists a document in the database (specifically, within the collection named after current HIT's identifier) with name matching the provided code. If it does, we retrieve that document and check whether the worker completed the experiment. If they did, their assignment is approved but the bonus must be determined. There are two options: (i) either they failed the comprehension check, in which case we also retrieve the bonus that they should be paid from the database (it had been computed previously based on the timings of answering comprehension questions), (ii) or they completed the game, in which case bonus is computed from their earnings in the game.

7.2.3 Challenges

Having given an overview of our design decisions and the way we interact with MTurk, we now describe the most notable challenges that we faced when developing the web application.

Performance

Recall that the time it takes for our tool to compute the decision of an agent increases exponentially with the lookahead of that agent. In a game such as trust game, assuming investor's endowment of four units and multiplication factor of two, agents have a choice of up to nine different actions. As a result, the decision tree becomes very large very quickly and it is impractical to set lookahead values greater than four. This is especially relevant for our experiment, since the time it takes for the bot to compute its action is the time participant spends waiting.

Our concern was that, if that time is too long, subject might get impatient and alter their strategy, thereby compromising results of our study. We have therefore committed to never exceed thirty seconds and keep it well below that limit most of the time. To achieve that, we experimented with various configurations of bot parameters to see what values of bot parameters we can afford. Naturally, the greater the bot's lookahead is, the more informed its decisions are. However, we have discovered that larger lookahead only makes a substantial difference when bot plays as investee. Intuitively, this stems from the fact the the role of investee in the trust game puts one at the mercy of the other player. Trust must be maintained at a reasonable level in order to sustain cooperation, but a short-sighted investee will be tempted to betray their opponent's trust by keeping most, if not all, of the money they receive. In the role of an investor, however, opponent's trust does not matter as much; what matters is investor's own trust towards their opponent.

As a result, we have identified that a lookahead of two, which enables the agent to consider their own transfer and opponent's return, is sufficient to compute optimal action in the role of investor. This allowed us to use more accurate meta-parameter estimations (as detailed in Appendix C). On the other hand, when bot plays as investee, we use a lookahead equal to four, as this was experimentally determined as the largest possible value that does not compromise performance. However, it comes at a trade-off of less accurate meta-parameter estimations.

WebPPL Immaturity

Recall that WebPPL is a fairly new probabilistic programming language. Its development kicked off in 2015, continued for the next three years and was mostly abandoned in early 2019. While we have not discovered any major bugs, certain aspects of the language, particularly error handling, leave a lot to be desired.

In the context of our web application, the main challenge was to run WebPPL code from Javascript. We have gathered this was possible based on online discussions that can be found in a webppl-dev Google group[1]. However, that feature is not documented and making it work necessitated studying the source code of the language.

This was made even more difficult by the fact that the primary method for running WebPPL code, `webppl.run`, performs compilation of the provided code prior to executing it. As it turns out, compiling WebPPL is remarkably inefficient; we have observed compilation times as high as ten seconds, though the average was roughly three seconds. In any case, time-sensitivity of our experiment demanded that this compilation time is drastically reduced.

A natural idea is to pre-compile the code, preferably as part of server initialisation, so that future runs can skip that step. Indeed, after further source code exploration, we identified appropriate methods that allowed us to separate the compilation and execution.

Worker Processes

When it is bot's turn to select an action in the trust game, the browser of the user playing the game queries the transfer amount by sending a request to our server. Now, a naive server implementation would, upon receiving such a request, make a call to a function (shipped as part of WebPPL library) that knows how to run WebPPL code and collect the result once this code has finished executing. The problem here is that, while the bot's action is being computed, which as we mentioned may take up to 30 seconds, the server is busy and cannot serve incoming requests. This might cause problems on the client-side too, with the application becoming unresponsive while the request is being processed.

As a result, Heroku places a hard, not configurable limit of 30 seconds on processing time of a request. In any case, it is recommended to keep it well under that limit, specifically under 500 ms. With that, all heavy computations that may take longer than that are run separately as part of so-called worker processes. This makes the whole operation of querying bot's action more complex. Rather

[1]`https://groups.google.com/g/webppl-dev`

than sending one request followed by receiving a response, the client now starts by sending an initial request. Upon receiving that request, the server kicks off the worker process and sends a response containing an identifier of that worker process to the client. This allows the client to repeatedly (every half a second in our case) query the server to check whether the worker process has finished computing bot's action. Until that is not the case, the server keeps sending a response indicating the action has not been computed. Once it is the case however, server responds with the computed action and the game can continue on the client side.

The way this is implemented on the server itself is with a queue. Every time a request arrives that demands bot's action to be computed, a job is created and placed in a queue. Worker processes that are not busy constantly scan that queue and, as soon a job is placed there, one of them will pick it up and start the computation. However, if all worker processes are busy when a new job arrives in the queue, that job must wait until a worker process becomes free (and possibly until all the jobs that are in front of it in the queue are processed). It is therefore important to choose the number of worker processes on the server appropriately, reflecting the expected number of simultaneous users of the application.

Game Description Wording

Even though trust game is a fairly simple scenario, explaining it concisely turned out to be challenging. It took many iterations of reviewing the game description followed by hearing feedback from people to arrive at the final text. In the end, we have opted for different descriptions depending on the role that the participant was assigned to, as detailed in Appendix C.

Custom Caching

One of the most useful features of WebPPL is the caching operator. When applied to a function, it memoises the return value every time the function is called, so that subsequent calls with the same arguments can return immediately by looking up the return value in the cache. Recall that our decision-making algorithm works by traversing the decision tree and computing various things, such as beliefs or utilities, at each node. Since many nodes are visited repeatedly, it is very useful to have the values already computed for a given node cached, so that future visits do not need to recompute those values.

This mechanism is particularly important when it comes to computing agent's belief, which is computed recursively and each recursive step, i.e., the belief update, is computationally expensive. Now, the problem is that the cache is only preserved for a single execution of a WebPPL program. However, a basic property of our multi-process server design is that every job may be completed by any worker process. Any data passed between processes must be serialisable, so storing the state of entire WebPPL program is highly impractical. As a result, every time a worker process starts processing a new job, no state from previously computed jobs for the same participant (which computed all actions of the bot up until this point) is available; in particular, the cache is lost.

We have run performance tests and determined that recomputing beliefs every time is costly, especially in the latter stages of the game. We have found that up to 50% of execution time of a given job was being spent computing bot's belief. We have therefore implemented custom caching, whereby the WebPPL program that computes bot's action also returns the belief cache (every cached function has a separate cache). We then process the cache to extract its selected entries and store them in a local Redis database. With that, every time a worker picks up a new job, it first retrieves the appropriate belief cache (identified by the ID of the user who requested this job) and passes it to the WebPPL program.

7.3 Results

Before describing our findings, we briefly mention how we carried out our experimental procedure in practice.

MTurk Task Setup Our aim was to obtain roughly a hundred data points, each provided by a participant who passes the comprehension check and completes the game. However, to test our web application and our procedure, we first ran a pilot study with eleven subjects. It went well overall, although only six out of eleven participants passed the comprehension check. This may have been partially due to the fact that we omitted the Masters qualification requirement.

Our estimation was that, in the actual study, failure rate would be no higher than 30%. We therefore set our HIT to accept up to 150 assignments. We also allowed two hours for each subject to complete the task. Of course, this is much more than the expected time of completion of our experiment. The reason for allowing participants this fairly generous amount of time to complete the task has to do with performance considerations. As detailed in Section 7.2.3, actions of the bot are computed by so-called worker processes. As soon as the number of users playing the game simultaneously is greater than the number of worker processes started on the server, performance of the application may suffer. Server cost is proportional to the number of worker processes. Our goal therefore was to minimise the number of simultaneous users at any time of the experiment.

Now, based on external advice, we had expected that the highest workload will take place in the first few hours once the HIT is published. Apparently, many MTurk workers run scripts that accept eligible assignments shortly after they are published. We reasoned that if we allow each participant two hours for completion, some will start the task as soon as they can, but others will do it twenty, forty or ninety minutes after accepting the assignment.

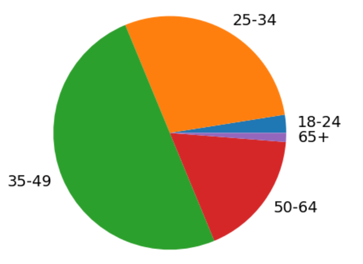
We set the lifetime of the task to be two days (48 hours). We had expected all available assignments to be completed within hours and regarded two days as a safe upper limit.

Outcome We now describe what actually happened upon publishing our task on MTurk. Accordingly with our expectations, there was a high level of activity immediately afterwards. Twenty MTurk workers accepted the HIT within a minute of its release; more than fifty completed the task in the first three hours of its existence. However, to our surprise, activity reduced considerably with time and we ended up with only 99 workers completing the task. Fortunately, success rate was higher than anticipated; n=80 participants passed comprehension check and gave us useful data. All the results presented below are based on those eighty subjects.

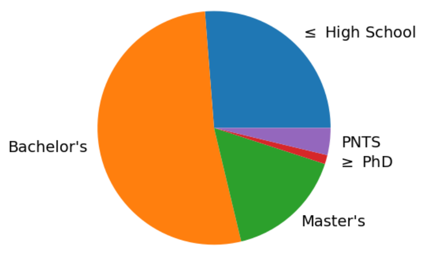
We find that the participants were predominantly middle-aged; half of them were aged between 35 and 40 years old and a further 23 were at least 25 years old but no older than 34 (see Figure 7.2a). More than half (48) were male (see Figure 7.2c). Most subjects had a university degree; majority Bachelor's (42) but there was one with at least a PhD (see Figure 7.2b). As is usually the case for MTurk studies, participants predominantly come from one of two countries, India or USA, the latter being four times more frequent than the former (see Figure 7.2d). Finally, most subjects reported limited (48) or fair (26) exposure to robotics (see Figure 7.3).

From the technical point of view, the experiment went well. One participant reported an application error preventing them from finishing the game. However, they were able to start again and complete the study upon refreshing the site. We paid them an extra $1 as bonus for the time lost. Inspection of the code of our application revealed that the error was caused by inappropriate handling of a particular edge case in which a worker process fails to compute the bot's action.

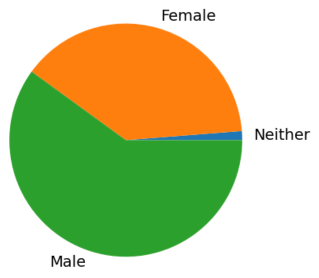
Recall that our aim was to keep bot's actions' computation time well below thirty seconds. We cannot claim we achieved this goal fully, as there were four instances where this limit was exceed. Not by much, however; the longest it took



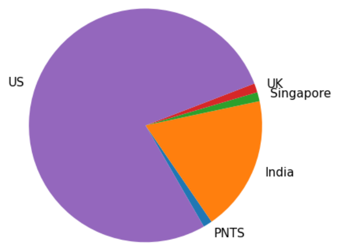
(a) Age



(b) Education



(c) Gender



(d) Nationality

Figure 7.2: Demographics of subjects (PNTS stands for “Prefer not to say”)

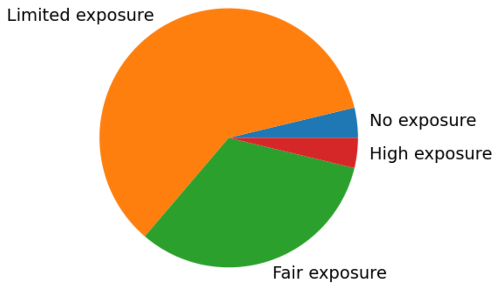


Figure 7.3: Exposure to robots of participants

the bot to compute its action was 35 seconds. On the plus side, in 99% of cases computation time was kept under 30 seconds. Moreover, in 90% of cases it was under 11 seconds and 50% of the time it was no more than 2 seconds.

With this preliminary information presented, we direct our attention to the objectives of our experiment.

7.3.1 Model Validation

We begin by analysing how behavioural predictions generated by our model fare against baseline predictors detailed in Section 7.1.3.

The overall comparison, for which all data from all subjects (regardless of treatment) were used, is depicted in Figure 7.4. Note that we summarise the prediction errors with a box plot with all data points overlayed on top of it. For every predictor, each yellow dot represents the mean squared error of behavioural predictions generated by this predictor for a participant. Since the game always consists of seven rounds, this MSE is an average of seven squared prediction errors (one prediction per round). Each prediction takes the form of a fractional number. Recall that probabilistic predictions, such as the ones generated by our model, are reduced to a deterministic, fractional prediction by taking an expectation. While this leads to some loss of information, this is the best single measure that unifies all predictors that we have found.

Besides individual data points, which are overlaid and technically not part of the box plot, our plots convey basic summary statistics of the data: each box represents the interquartile range (IQR), i.e., it contains the middle 50% of MSEs when they are ordered. The whiskers extend to the furthest point (both ways) that is no further than 1.5 times the IQR distance (i.e., the length of the box) from box edges. In other words, the whiskers show the minimum and maximum values in the data set excluding any outliers. For example, the box plot for the UNIFORM predictor in Figure 7.4 has no outliers and, indeed, whiskers are located at the minimal and maximal mean squared errors. Finally, two measures of central tendency are shown: the median is represented by a red line across the box whereas green triangles indicate means.

Now, returning to Figure 7.4, we can see that DANG outperforms our model in terms of median prediction error ($\mathrm{M}_{\text{DANG}} = 1.58$ compared to $\mathrm{M}_{\text{CSMG}} = 1.84$ for CSMG) and very slightly on the mean too ($\mu_{\text{DANG}} = 1.82$ vs. $\mu_{\text{CSMG}} = 1.92$). On the other hand, we note somewhat smaller standard deviation of our predictor ($\sigma_{\text{CSMG}} = 1.42$ vs. $\sigma_{\text{DANG}} = 1.44$). We also observe that the highest MSE for DANG is 10.54 compared to the maximum of 7.14 for our model. Other baseline predictors performed significantly worse than DANG; UNIFORM is consistent (no outliers) but not very accurate, while AVERAGE and LAST gave similar results. All the medians, means and standard deviations are summarised in Table 7.1.

We also perform the exact same analysis on certain subsets of our data. In particular, we first restrict our attention to plays in which our bot used participants' answers in the pre-game questionnaire to generate an informed prior (i.e., the

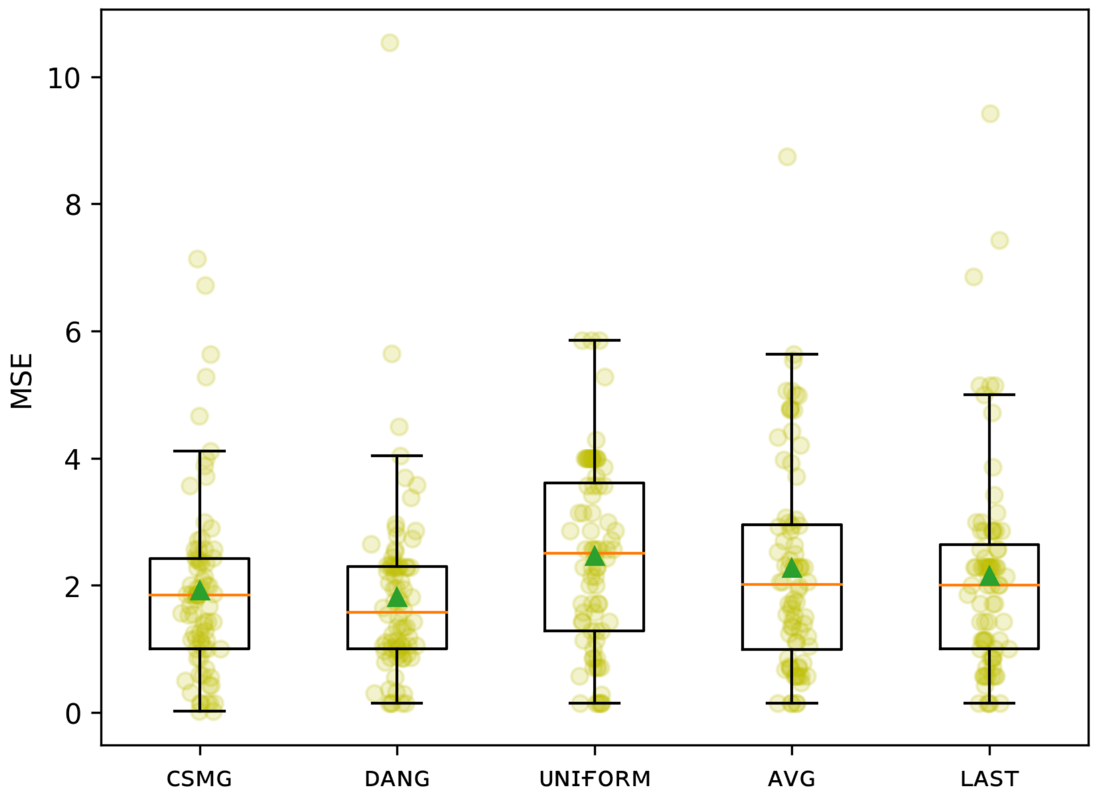


Figure 7.4: All predictors compared on all data points. Each yellow dot represents the MSE of seven predictions generated for some participant.

Table 7.1: Summary statistics (mean, median, standard deviation) of predictions generated by all predictors on all data ($N = 80$)

	μ	M	σ
CSMG	1.92	1.84	1.42
DANG	1.81	1.57	1.44
UNIFORM	2.56	2.5	1.45
AVERAGE	2.27	2.02	1.67
LAST	2.15	2	1.66

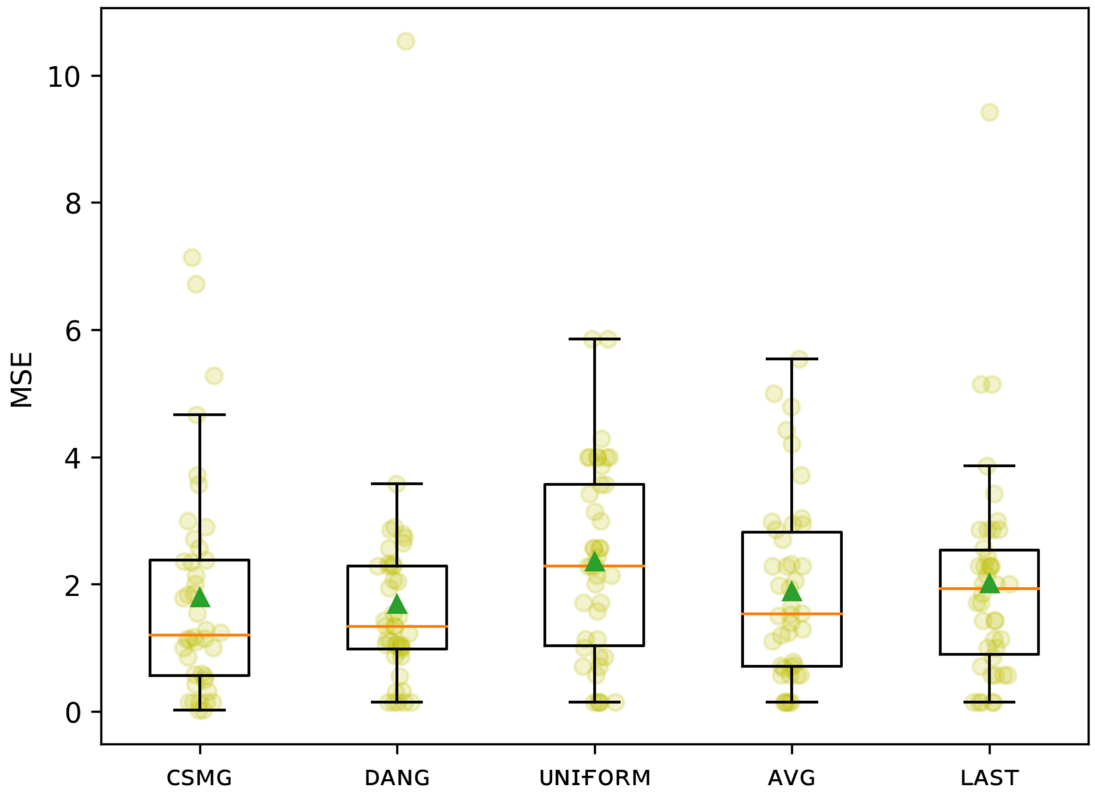


Figure 7.5: All predictors compared on INFORMED treatment ($N = 42$). Each yellow dot represents the MSE of seven predictions generated for some participant.

Table 7.2: Summary statistics (mean, median, standard deviation) of predictions generated by all predictors in INFORMED treatment

	μ	M	σ
CSMG	1.79	1.2	1.71
DANG	1.69	1.34	1.66
UNIFORM	2.36	2.28	1.53
AVERAGE	1.89	1.53	1.44
LAST	2.01	1.93	1.7

INFORMED treatment). The box plot generated based on this subset of our data is displayed in Figure 7.5, with summary statistics presented in Table 7.2.

Interestingly, all predictors show an improvement in the INFORMED treatment compared with all data, suggesting that, by chance, the behaviour of participants' allocated to that treatment was more predictable. However, the improvement is most pronounced for our model, particularly in terms of median prediction error. This comes at a price of increased standard deviation though, as the figure clearly shows.

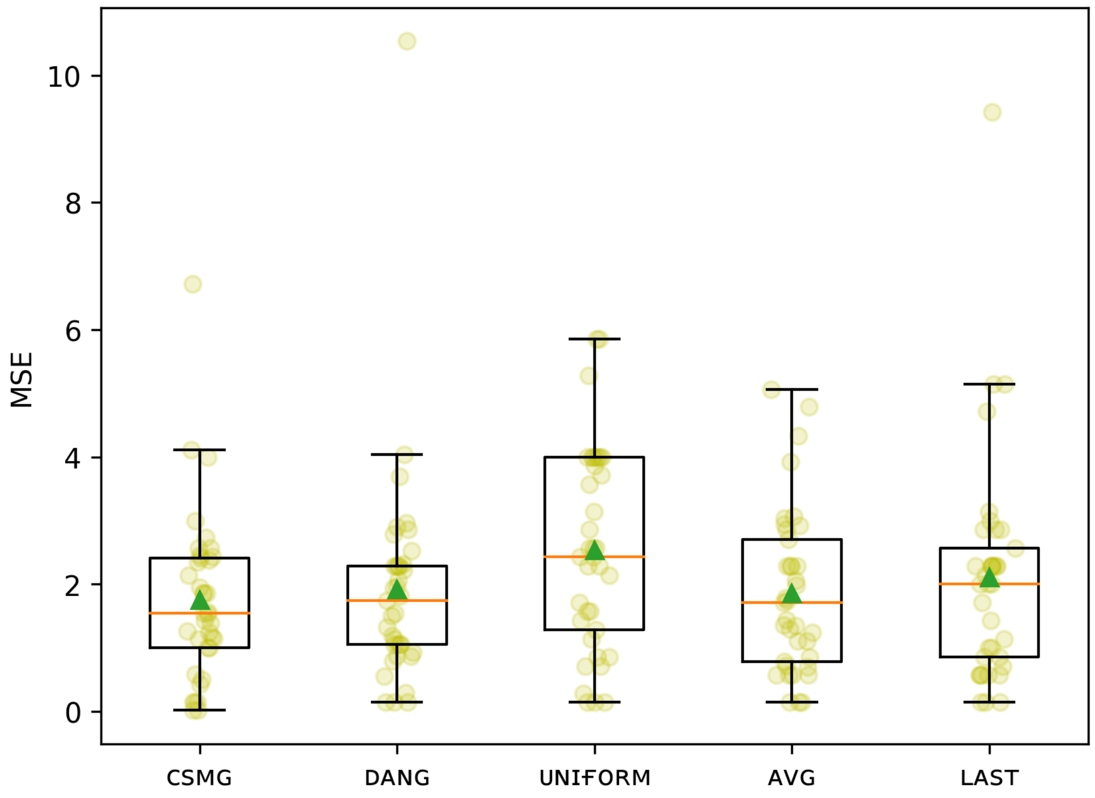


Figure 7.6: All predictors compared on UNDISCLOSED treatment ($N = 37$). Each yellow dot represents the MSE of seven predictions generated for some participant.

Table 7.3: Summary statistics (mean, median, standard deviation) of predictions generated by all predictors in UNDISCLOSED treatment

	μ	M	σ
CSMG	1.75	1.54	1.32
DANG	1.92	1.74	1.75
UNIFORM	2.54	2.43	1.59
AVERAGE	1.86	1.71	1.29
LAST	2.1	2	1.79

Another treatment we focus our attention on is UNDISCLOSED, characterised by the fact that subjects were unaware of the number of rounds that would be played. This is a standard measure taken (notably by Dang et al. [121]) to avoid participants changing their behaviour in the last round. We therefore hypothesised predictions would be better for this cohort. The results are displayed in Figure 7.6 and Table 7.3.

Indeed, four of five predictors improve under the UNDISCLOSED treatment compared to all data, both in terms of means as well as medians of prediction

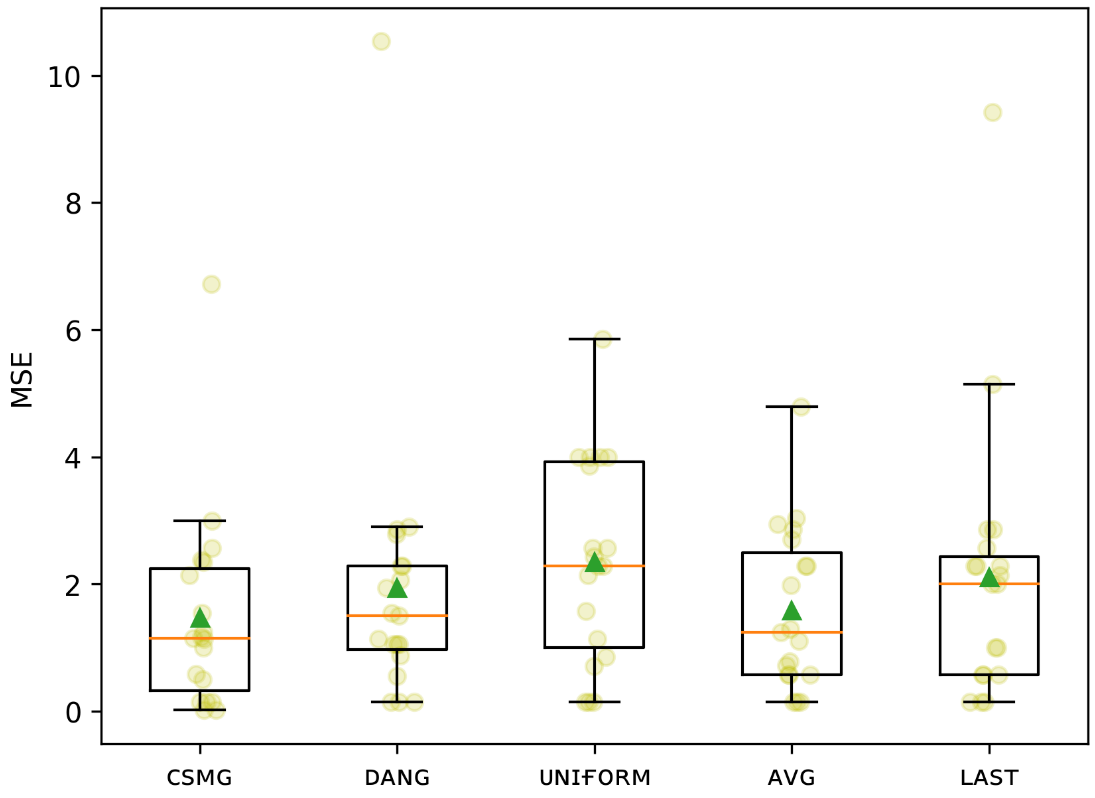


Figure 7.7: All predictors compared on an intersection of UNDISCLOSED and INFORMED treatments ($N = 19$)

errors. However, DANG scores worse on both measures; in fact, it is outperformed by AVERAGE under this treatment.

Finally, we consider a final, smallest subset of our data, formed as an intersection of INFORMED and UNDISCLOSED treatments. This subset can be characterised as providing optimal conditions for our predictor to perform well – not only is a prior over participants' preferences and cognitive abilities available, but also subjects are not tempted to act opportunistically at the end of the game.

Indeed, the box plot in Figure 7.7 and summary statistics in Table 7.4 demonstrate the superiority of our predictor under these conditions, with AVERAGE coming in close second place, surprisingly again outperforming DANG.

Overall, only CSMG outperforms the three baseline predictors under all data subsets and shows consistent improvement under treatments that should theoretically favour it. However, due to relatively small sample size, no statistically significant conclusions can be drawn regarding the superiority of CSMG over DANG or vice versa.

Table 7.4: Summary statistics (mean, median, standard deviation) of predictions generated by all predictors in the intersection of UNDISCLOSED and INFORMED

	μ	M	σ
CSMG	1.47	1.14	1.58
DANG	1.94	1.5	2.27
UNIFORM	2.35	2.28	1.62
AVERAGE	1.59	1.25	1.28
LAST	2.1	2	2.17

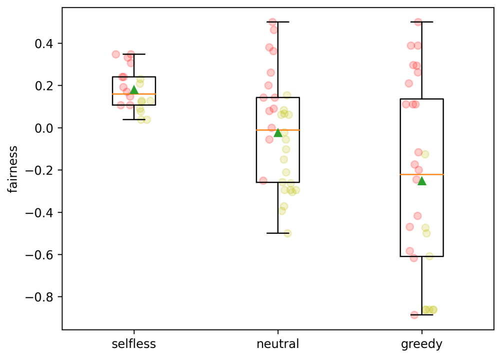
7.3.2 The Effect of Bot Parameters on Game Outcomes

The second major result of our experiment concerns the differences in game outcomes depending on the "character" of the bot. Recall that we designated three different "bot personalities" by varying goal coefficients: SELFLESS characterised by a vector $\langle .3, .7 \rangle$, NEUTRAL marked by goal coefficients $\langle .5, .5 \rangle$ and GREEDY with $\langle .8, .2 \rangle$. Intuitively, the first element of the vector reflects bot's relative attachment to money, while the second one quantifies the relative importance of maintaining a relationship with their human opponent.

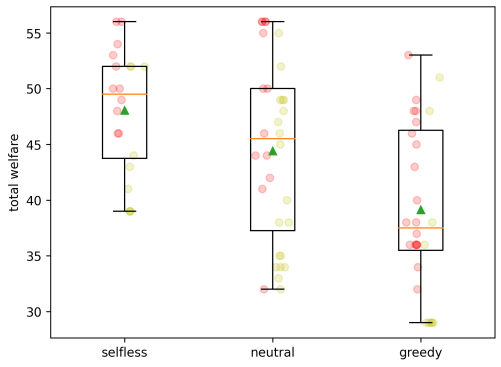
The first set of results is displayed in Figure 7.8; again, we use box plots with individual points overlaid to visualise the data. Recall from Section 7.1.2 that we quantify game outcomes using two measures: (i) fairness, which captures relative difference in earnings of both players (in this case, positive values indicate that the human earned more, and vice versa), and (ii) total welfare, which describes the sum of earnings of both players. Note that we use two different colours for the data points, to distinguish plays where bot is an investor (in yellow) or an investee (in red).

Looking at fairness first (Figure 7.8a), a few interesting observations can be made. First of all, means, medians and standard deviations are markedly different, summarised in Table 7.5. The second remark worth making concerns the skewness of the GREEDY data towards negative values, which shows that the greedy bot was quite successful at maximising its profit. Worth noticing also is small standard deviation in the SELFLESS cohort and near perfect average fairness of NEUTRAL plays.

To investigate if the differences between group means are statistically significant, we perform the Mann-Whitney U rank test. We use a non-parametric method since group variances differ significantly and the normality of our data is doubtful (as evidenced by Shapiro, D'Agostino and Kolmogorov-Smirnov normality tests), particularly in the GREEDY cohort. We obtain $p = 0.00157$ when comparing



(a) Fairness



(b) Total welfare

Figure 7.8: How game outcomes depend on bot's character (yellow points correspond to plays in which bot is an investor, red points represent bot as an investee)

Table 7.5: Summary statistics (mean, median, standard deviation) of fairness depending on bot's character

	μ	M	σ
SELFLESS	0.18	0.16	0.1
NEUTRAL	-0.02	-0.01	0.26
GREEDY	-0.25	-0.22	0.46

SELFLESS with NEUTRAL, $p = 0.00327$ when comparing SELFLESS with GREEDY and $p = 0.0841$ when comparing NEUTRAL with GREEDY. This suggests high significance in the first two comparisons, even with Bonferroni correction applied, which is needed as we perform three comparisons. In particular, since in both cases $p < 0.0033 = \frac{0.01}{3}$, Bonferroni inequality yields that the experimentwise Type I error rate α_E is at most 0.01.

Looking at total welfare next (Figure 7.8b and Table 7.6), measures of central tendency again differ manifestly. Mann-Whitney U rank test this time shows

Table 7.6: Summary statistics (mean, median, standard deviation) of total welfare depending on bot's character

	μ	M	σ
SELFLESS	48	49.5	5.63
NEUTRAL	44.4	45.5	8.09
GREEDY	39.1	37.5	7.41

very high significance in the SELFLESS vs GREEDY comparison, with $p = 0.00008$, meaning that the likelihood of committing Type I error is less than 0.1%. On the other hand, the two other comparisons yield much higher p values; $p = 0.109$ for SELFLESS vs NEUTRAL and $p = 0.0227$ for GREEDY vs NEUTRAL.

Apart from game outcomes, which are highly objective, we also investigate how participants' subjective impressions are affected by various bot types. Recall that after the game each participant answers three questions. The first of those asks the subject to judge how human the bot's behaviour felt on a 5-point Likert scale. The other two questions concern change in participant's trust – to the bot that they just faced and to robots in general, respectively.

The answers to the first question are summarised in a bar plot of Figure 7.9. Note that, due to difference in group sizes, we use relative, rather than absolute, measures of frequencies of answers. Therefore, the units of the y axis are percentages and each bar should be interpreted as the percentage of all participants in a given group (SELFLESS, NEUTRAL or GREEDY) that selected a given answer. The differences between groups are not significant, but it is worth noting that NEUTRAL was judged the least human by the participants.

The distribution of the answers to the other two questions is depicted in Figure 7.10. We also visualise the correlation between the two responses using a confusion matrix, displayed in Figure 7.11. Each of the nine squares of the grid represents a combination of answers and the value inside the square is the number of times such combination was observed. For example, the square marked with a nine in the left-bottom corner counts cases where a participant reported a decrease in trust towards the bot as well as towards robots generally. The bottom left to top right diagonal contains the combinations where both answers match.

7.3.3 Prior Effect

The final major result that we present investigates how the quality of predictions generated by our model depends on whether an informed prior is utilised. We have already seen in Section 7.3.1 that our predictor performs better when that is the case. Presently, we make those observations more rigorous by focusing on a comparison of the prediction error of CSMG between the two treatments, INFORMED and UNIFORM.

The results are plotted in Figure 7.12. We note the presence of more outliers in the INFORMED group, and the outliers present are more extreme – we attribute this to chance (if a participant behaves unexpectedly then even a good prior will generally not make much of a difference). Despite that, mean prediction error is smaller for INFORMED – $\mu_{\text{INFORMED}} = 1.79$ vs $\mu_{\text{UNIFORM}} = 2.07$. However, the

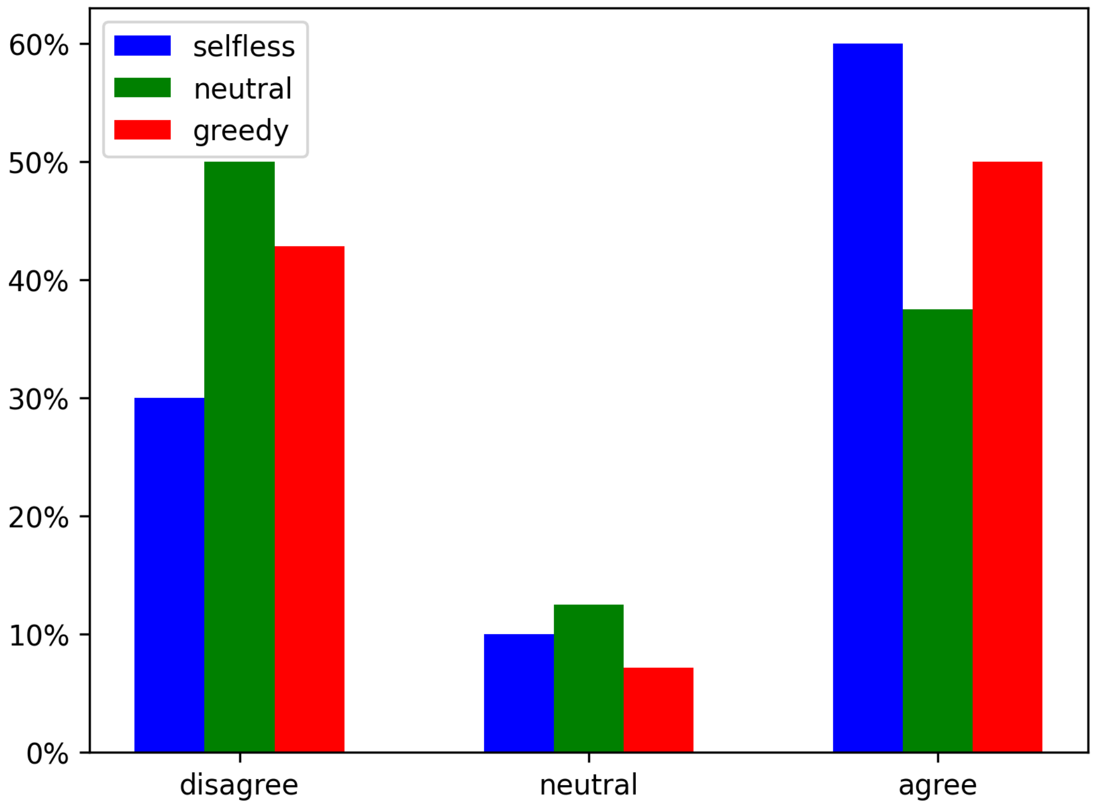
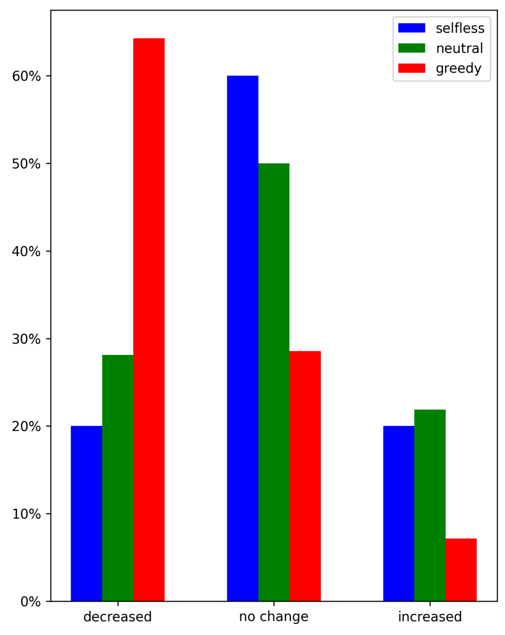


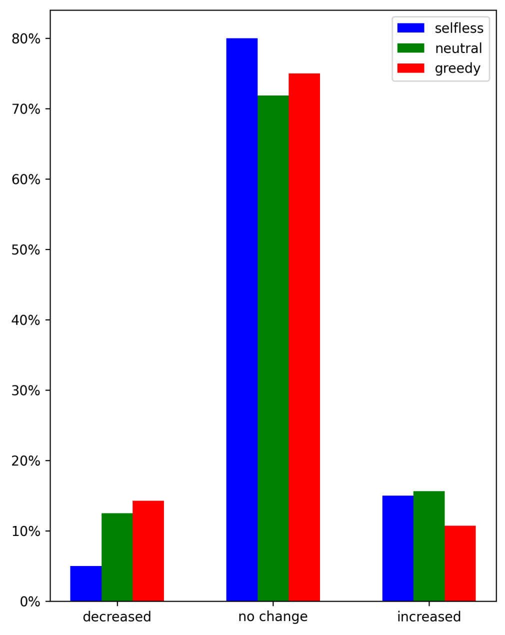
Figure 7.9: Distribution of answers to the question "Do you agree bot played like a human?", categorised by bot character

difference is much more pronounced if we consider the medians – $M_{\text{INFORMED}} = 1.2$ vs $M_{\text{UNIFORM}} = 1.86$. Even more remarkable is the fact that the smallest mean squared error recorded for UNIFORM is 0.43, which is greater than the 19th percentile (0.40) in the INFORMED cohort. In other words, a fifth of predictions generated using the informed prior are better than the best prediction generated using the uniform prior. In fact, 20th percentile for UNIFORM (1.43) is significantly larger than the median for INFORMED. Indeed, Mann-Whitney U rank test reveals a significant difference between the two cohorts ($p = 0.0397$), which is even more impressive considering relatively small sample sizes ($N = 42$ for the UNIFORM group).

Of course, we have seen that predictions of our model are significantly better for the UNDISCLOSED cohort. To eliminate the possibility of confounding our result, we performed the same analysis restricted to plays in which participants did not know the game's horizon. The results are depicted in Figure 7.13. Interestingly, nearly all outliers in the INFORMED group disappear, bringing the mean down ($\mu_{\text{INFORMED}} = 1.47$) while the median remains low ($M_{\text{INFORMED}} = 1.14$). Improvement in the UNIFORM cohort is negligible – $\mu_{\text{UNIFORM}} = 2.05$, $M_{\text{UNIFORM}} = 1.85$. As a result, despite



(a) Distribution of answers to the question "How did your trust to the bot change after playing the game?", categorised by bot character



(b) Distribution of answers to the question "How did your trust to robots in general change after playing the game?", categorised by bot character

Figure 7.10: Answer distributions for trust-related questions

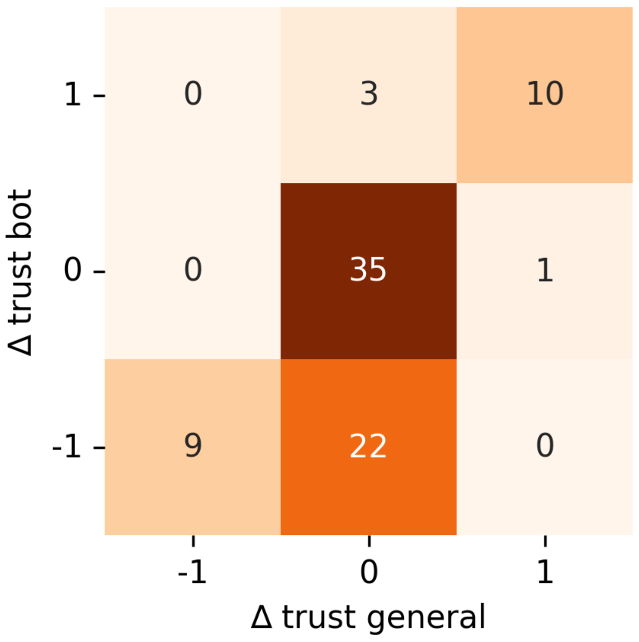


Figure 7.11: Confusion matrix showing how change of trust towards the bot (y axis) translates into trust change towards robots generally (x axis). Values -1, 0 and 1 represent participant reporting a decrease, no change and an increase of trust, respectively

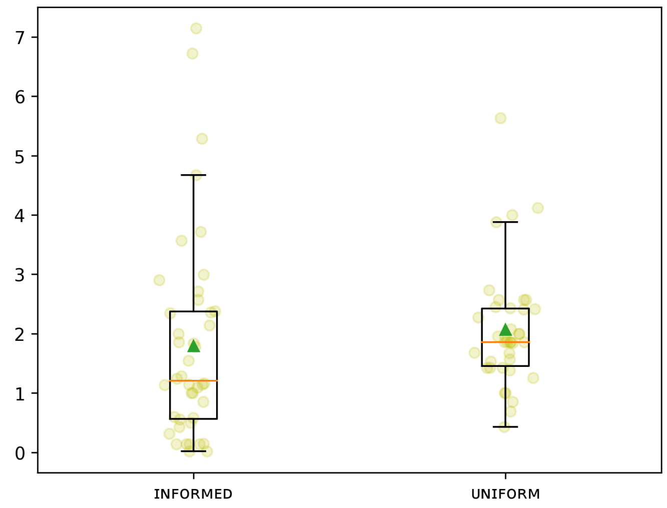


Figure 7.12: Prediction errors of CSMG predictor categorised by type of prior used; all data points

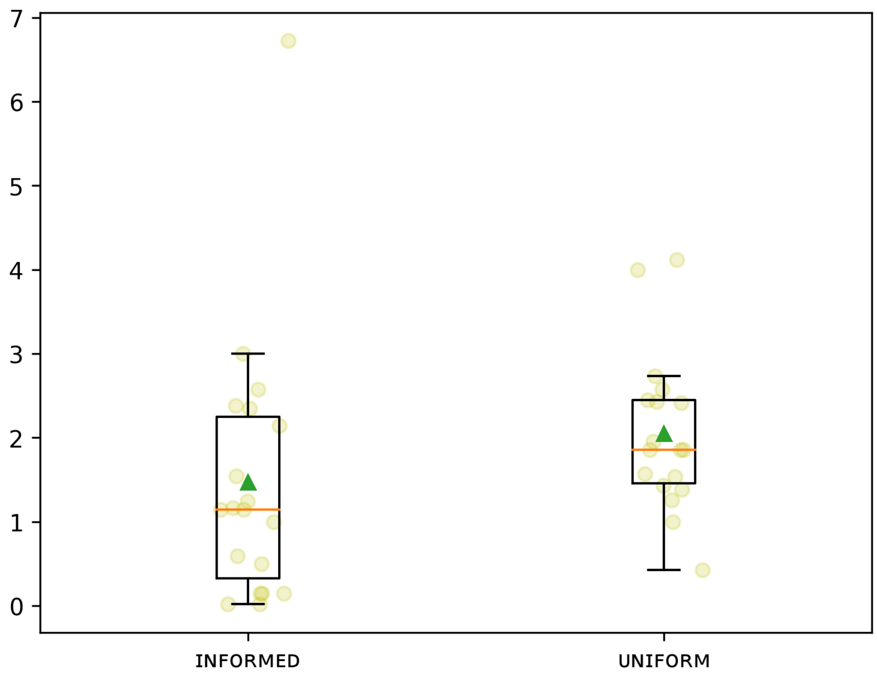


Figure 7.13: Prediction errors of CSMG predictor categorised by type of prior used; restricted to UNDISCLOSED cohort

reduced sample size, the difference between group means became more significant by restricting data to UNDISCLOSED – Mann-Whitney U rank test gives $p = 0.0275$.

This confirms our hypothesis that an accurate prior can improve the quality of behavioural predictions and falsifies a statement by Dang et al. [121], who claimed that “the only reliable available information for predicting users behavior is their behavior during previous transactions”.

7.3.4 Minor Results

Having described the main results of our experiment, we now briefly describe some findings that are not directly related to our framework, but provide insights into human nature and our relationship to robots.

Human Opportunism We begin by resolving the hypothesis claiming that telling subjects how many rounds the game will last alters their behaviour in the last round of the game. To investigate it, we focus on plays where the participant played as investee and where the bot invested a non-zero amount in the last round (which happened in a little over half of the cases). Also, rather than considering the absolute number of units returned, we focus on the proportion of the amount invested by the bot that was returned by the subject. Recalling that every investment is doubled in transit, such a proportion falls in the range between 0 and 2. Figure 7.14 depicts the frequencies of various return proportions, split into four buckets to aid readability. Despite a very small sample size ($N = 20$ plays satisfy the condition mentioned above), the trend is clear, with the vast majority of humans who knew how many rounds will be played deciding to defect in the last round. This is confirmed by the Mann-Whitney U rank test, which suggests the difference in means between groups is significant ($p = 0.0304$), thereby confirming our hypothesis.

11-20 Money Request Game Since we used the 11-20 Money Request Game to estimate lookaheads of participants, we report the answers we collected and compare them to what the authors of the game reported as part of the original study. This comparison is depicted in Figure 7.15.

Effectiveness of Questionnaire Time Controls Recall that we have implemented a time control mechanism whose goal was to force participants to carefully choose their answers to questions in the pre-game questionnaire. This mechanism was described in some detail in Section 7.1.5 but the gist of it is as follows: the questions, rather than appearing all at once, appear one by one with a time delay that differs for each question, reflecting its difficulty (and thereby expected time needed to answer it).

To evaluate the effectiveness of this measure, we have recorded time series of subjects' interactions with the questionnaire. Each time series consists of a sequence of timestamped events. The first event in the sequence is always the `start` event, which is triggered when the participant starts the questionnaire after reading (or not) the introduction. After that, every time the subject selects an answer to

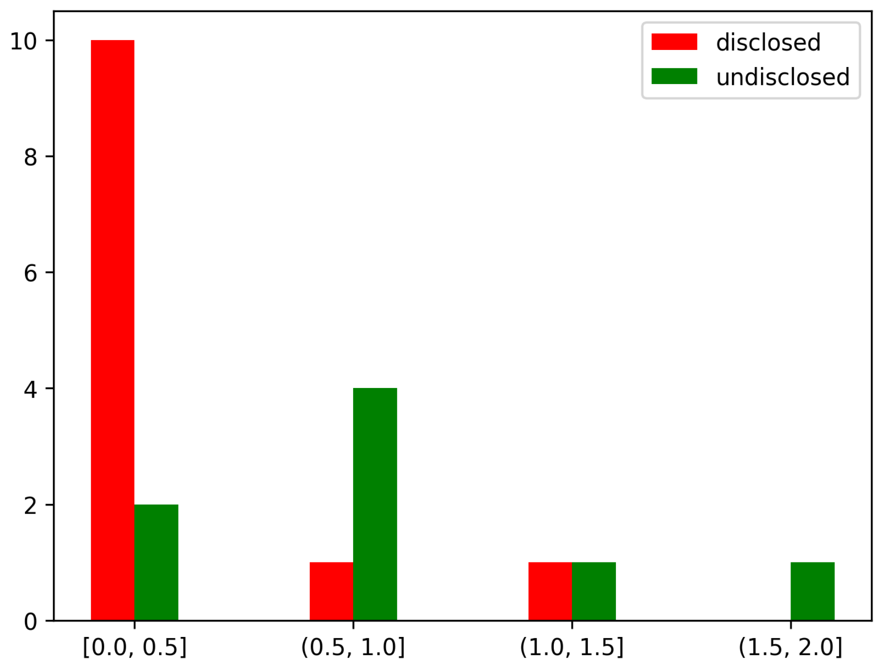


Figure 7.14: Number of subjects returning a given proportion of money received in the last round of the game, categorised by whether game horizon is disclosed to the participant

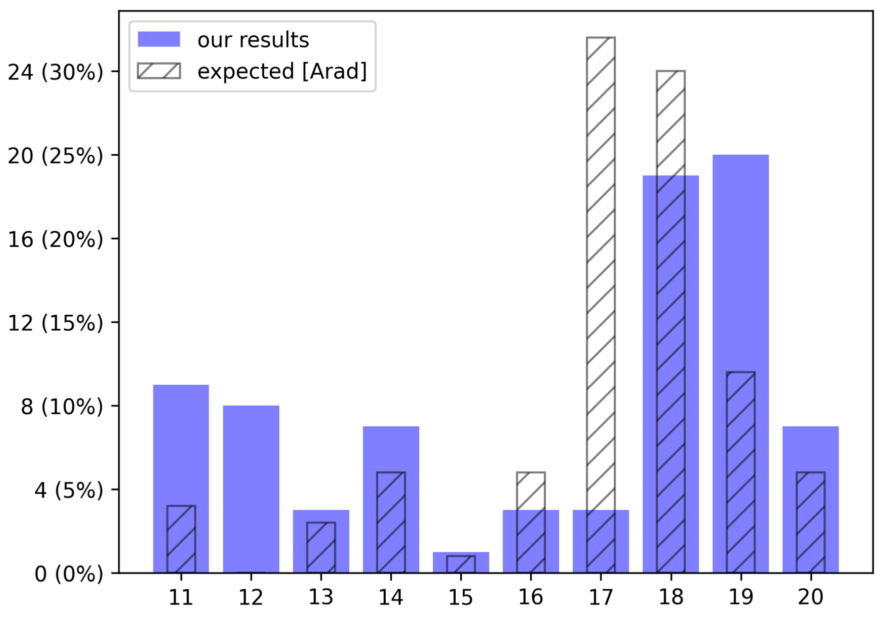


Figure 7.15: Distribution of answers to the 11-20 Money Request Game observed as part of our study (in blue) with a backdrop of the distribution expected based on results from Arad et al. [122] (diagonal hatch)

Table 7.7: Time allocations (in seconds) for each question

Q1	Q2	Q3	Q4	Q5	Q6
25	15	20	25	10	10

a question, or changes previously selected answer, a new (timestamped) event, named after that question (e.g., `moneyRequest` or `lottery1`), is added to the time series. Note that, if a participant interacts repeatedly with one question without interacting with other questions in between, we save this as one event timestamped at the last interaction. The last event is always `submit`, with a timestamp that represents the total time spent answering the questionnaire.

The way we analyse this time series data is a combination of quantitative and qualitative methods. As part of the latter, consider Figure 7.16, which we will use to illustrate how we evaluate the effectiveness of our mechanism. We will refer to individual plots from that Figure as (r, c) where $1 \leq r \leq 6$ is the row number and $1 \leq c \leq 4$ identifies the column. Table 7.7 presents time allocations for each question, defined as the time that elapses between this question being first shown and the subsequent question being displayed (or, for Q6 which is the last question, it is the time that elapses between displaying the question and activating the submit button). Note that the time allocation is designed to act only as the lower bound on the time subject spends contemplating the answer – an answer may be selected or changed even once this time elapses.

Now, what we would like to see in plots from Figure 7.16 are intervals whose lengths reflect the allocations from Table 7.7 with at least ten seconds spent reading the introduction. For example, plot $(5, 2)$ is a great example of a fairly efficient subject with exemplary time allocation, $(4, 1)$ suggests another participant who paid attention and gave some thought to the questions, albeit somewhat less efficiently. Other examples are $(2, 3)$, $(6, 1)$, $(6, 2)$.

What we do not want to see are less uniform timelines, particularly those where a significant proportion of time is spent on Q1. This suggests a participant who starts the questionnaire, goes off to do something else while running down the clock and more questions to appear, and then comes back, answers the questions quickly and proceeds to the game. This is made even worse if combined with negligible reading time. Fortunately, no plot in Figure 7.16 fits this bill perfectly, but $(1, 3)$, $(3, 3)$ and $(5, 4)$ are suspicious.

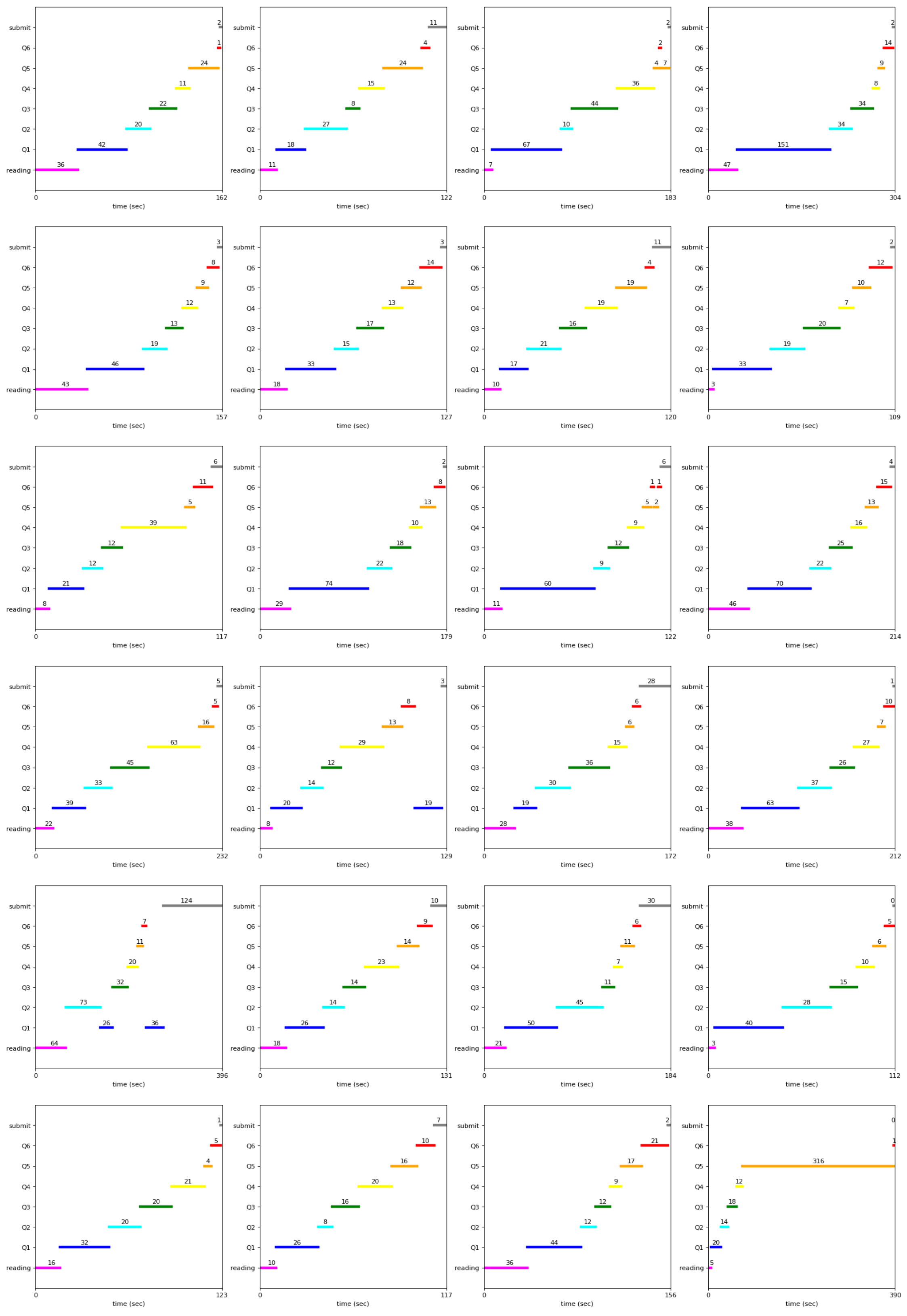
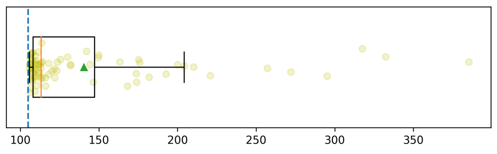
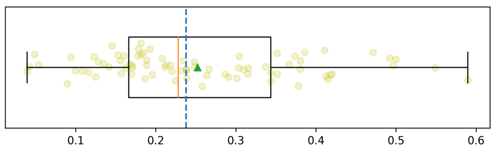


Figure 7.16: Chronology of completing pre-game questionnaire for a random sample of twenty-four participants



(a) Total questionnaire completion times of all participants with reading time excluded; dotted line marks the theoretical minimal completion time



(b) Proportion of time participant a takes to answer the first question relative to total questionnaire completion time (reading time excluded); dotted line indicates expected proportion

Figure 7.17: Quantifying questionnaire completion times

This qualitative analysis, as part of which we studied questionnaire timelines of all participants, suggests that our time control was fairly effective. This conclusion is further scrutinised using data. In particular, Figure 7.17a shows the distribution of total times spent by subjects completing pre-game questionnaire (where time reading the introduction is excluded). The dotted line marks 105 seconds, which is the sum of time allocations for each question and the theoretical minimal completion time. We report the median completion time of 113 seconds and mean completion time of 140 seconds. Therefore, a sizeable proportion of participants used more time than the bare minimum.

Next, Figure 7.17b displays the ratios of time a participant takes before answering the first question to the total questionnaire completion time (reading time excluded again). The dotted line indicates the expected ratio, based on time allocations from Table 7.7. Indeed, even though the deviation is significant ($\sigma = 0.12$), both the median (M $=$ 0.228) and the mean ($\mu = 0.252$) are highly consistent with the expectation.

7.4 Discussion

We now proceed with a discussion of the results just presented. We primarily focus on interpreting the major results of our study but we briefly mention some of the minor findings too.

Evaluating Predictors We begin by noting an important difference between the two main contenders: CSMG – the predictor based on our model and DANG – a trust-based predictor specially designed for forecasting behaviour in repeated trust game. Predictions generated by CSMG are probabilistic, offering not just the forecast but also an indication of confidence in its accuracy. Moreover, the predictions are explainable by the model that underlies them. That same framework can be applied to many other scenarios, such as the ones covered in Section 6.2, and generate behavioural predictions in those novel settings.

On the other hand, DANG is based on intricate mathematical formulations and recurrence relations designed to capture certain behavioural patterns hypothesised to occur in humans. The method does not generalise outside of the trust game. Moreover, the predictions generated by DANG are based on a real number between 0 and 1 which represents trustworthiness of an agent. The expected transfer is computed as a product of the maximal possible transfer and this trustworthiness value, most commonly resulting in a fractional number. In this particular context, such a prediction is fairly reasonable and, most importantly, its error can easily be computed. However, one may also argue that a deterministic behavioural prediction should be limited to choosing a single action from those available to an agent.

In any case, it is not our role to arbitrate on such matters. However, in order to quantitatively compare the two predictors, we had to settle on a single representation. To enable fair comparison, we decided to conform our predictor to the format of DANG, thereby losing all the extra information that probabilistic predictions carry. Admittedly, it is also much more straightforward to convert our probabilistic prediction to a deterministic one (as long as fractional forecasts are allowed) than vice versa.

Keeping all this in mind, the fact that our predictor comes so close to DANG when evaluated on all the data points should be considered a success. However, as shown by Figure 7.14, human behaviour differs greatly between the UNDISCLOSED and DISCLOSED treatments (the figure is limited to human behaviour as investee, but their behaviour in the role of investor is affected too). Now, we could have programmed our model and the bot so that it knows the game horizon in the DISCLOSED condition, which would have resulted in even better predictions. However, we decided to instead keep the bot naive so that it does not alter its behaviour towards the end, enabling us to properly test our hypothesis regarding human opportunism. As a result, it is to be expected that, under the disclosed horizon, CSMG predictions will be less accurate, particularly in the latter rounds of the game. Indeed, it should come as no surprise that if we consider ten participants for whom CSMG predictor performed the worst, we find that seven cases belong to the DISCLOSED treatment.

Restricting our analysis to UNDISCLOSED treatment effectively gets rid of cases when the bot is under-informed. However, if we combine this with a restriction to the INFORMED treatment, we are then focusing on cases when the bot is over-informed, where the extra information is used to improve the accuracy of bot's prior over the participant. Informed prior gives our model an edge over its competitors,

Table 7.8: An observed play, in which participant played as an investor. Each column represents one round of the game, consisting of a subject's investment in the first row, followed by a bot's return in the second row.

	R1	R2	R3	R4	R5	R6	R7
subject	4	4	4	4	3	4	2
bot	0	0	0	0	0	0	0

which allows it to produce extremely accurate predictions in some cases. In fact, out of ten plays on which our predictor performed best, nine were under informed prior. If we zoom out and consider top twenty plays, fifteen of them belong to the INFORMED treatment.

Another reason why our predictor performs so well in the intersection of UNDISCLOSED and INFORMED treatments is the complementarity of the improvements that both restrictions bring about. As argued above, restricting to UNDISCLOSED means participants' behaviour is more predictable in the latter stages of the game. On the other hand, informed prior improves predictions in the initial rounds where no, or little, behavioural data is available.

We finish our discussion of predictors by taking a closer look at a particular play that illustrates well the difference between CSMG and DANG. In fact, the play under consideration is the one where our predictor fared worst ($\text{MSE}_{\text{CSMG}} = 7.14$), but DANG did quite well ($\text{MSE}_{\text{DANG}} = 1.35$). The participant played in the role of an investor and they knew how many rounds the game would last for. Table 7.8 displays the sequence of actions that occurred.

Now, our model keeps predicting that the next action of the participant will be a low transfer. Initially, this stems from a prior that was inaccurate in this case, as it assumed low trust of the human. As the game progresses, the prediction does not change significantly, since it is mostly driven by the assumption that uncooperative behaviour of the bot must have decreased human's trust further. In this case, resilience (or irrationality) of the human was stronger than anticipated by the bot. On the other hand, DANG's predictions are highly accurate in this case as they are generated solely based on past behaviour of the human (i.e., bot's behaviour is not taken into account).

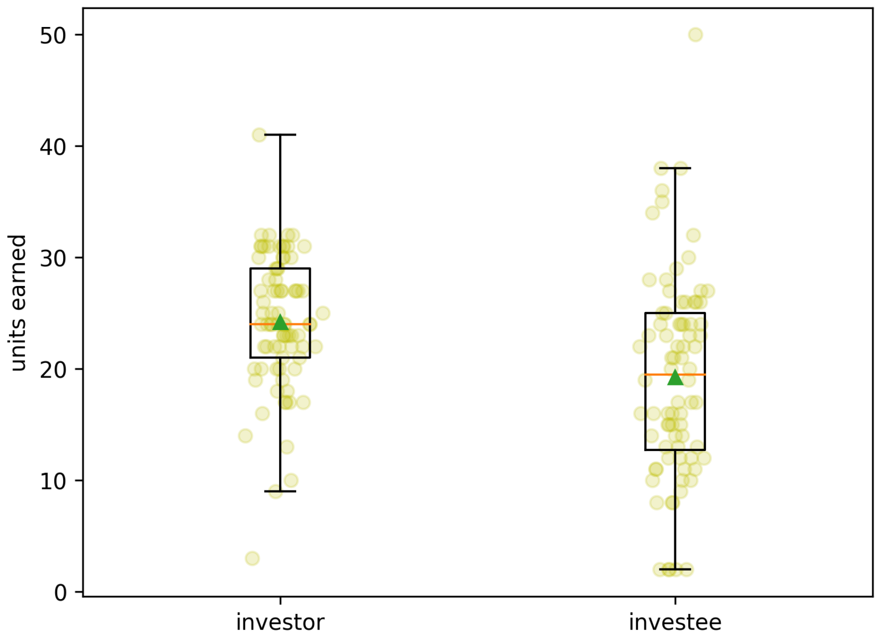


Figure 7.18: Comparison of total earnings in the game by role

Bot Character When it comes to the results comparing game outcomes between various bot types, we have learned that goal coefficients indeed influence game outcomes to a great extent. Considering fairness first, the difference in variances between bot types is quite remarkable. Selfless bot was designed with generosity in mind, but upon encountering repeated uncooperative human behaviour, it eventually loses its patience and stops being generous. As a result, games involving a selfless bot are usually one of two types: (i) both players being nice to each other, which leads to high investments and high total welfare, and (ii) human being noncooperative, which makes the bot noncooperative after a few rounds and results in lower social welfare. Interestingly, case (i) is much more common when human plays as investor while case (ii) occurs more often with the bot investing and the human trying to outsmart it. Anyhow, both cases end up producing games where human earns more, but not much more.

Neutral bot cares a little more about the money and a little less about human's trust, compared to the selfless bot. As a result, it loses its patience faster and punishes noncooperative participants earlier and more severely, resulting in a number of games in which the bot out-earns the human. These games are also characterised by low total welfare, explaining its higher variation in this cohort. This is partly because the bot does not give second chances – once it stops trusting the human, it does not give him or her an opportunity to regain its trust.

That brings us to the final bot type, the greedy one, which stretches the range of outcomes further. Greedy bot is primarily concerned with maximising profit and achieves this goal remarkably well. This is true despite a fairly unfavourable split of roles – out of twenty-eight plays that feature the greedy bot, only nine have it play as an investor. However, as Figure 7.18 shows, investors generally earn more, reflecting their commanding role in the game (in fact, investee is at investor's mercy to earn anything at all).

The greedy bot approaches the game as follows: as an investor, it is very careful with its money, investing high only if it trusts the human a lot. This produces a median earning of twenty-seven units, compared to the human's two. The difference in means is smaller but still very significant: $\mu_{bot} = 28.6$ vs. $\mu_{human} = 6.6$. But perhaps more remarkable are the results when the bot plays as investee. In that role, due to its short-sightedness (a consequence of performance limitations) the bot plays quite opportunistically, particularly when faced with high investments. This turned out to be a successful strategy as many humans, perhaps in disbelief of such uncooperative behaviour of the bot, kept investing the money despite receiving no returns. In one instance, the bot earned a whopping fifty units, far exceeding the maximum earning of any human (which was thirty-two units). In fact, on five occasions, the bot earned more than any human has. This can be linked to studies pointing out humans' tendency to overtrust automation [75, 123].

Overall, these results show that our model is capable of capturing a variety of personalities that result in a range of behaviours with simple manipulations of goal coefficients. We have observed that being generous in trust game pays off in the long term, but only if both players act that way. It is also quite remarkable that out of forty-three participants who played the game as investors, only one invested no units in the first round. While this may partly be explained by human curiosity and the desire to maximise profits, it also shows that humans generally trust, and often overtrust, robots.

We also briefly comment on the results from the post-game questionnaire. First of all, we believe that behavioural data is more reliable and should be taken more seriously than the post-game questionnaire. Nevertheless, certain interesting conclusions may be drawn from participants' answers. Regarding perceived "humanness" of the bot, the results are not significant, but suggest that the behaviour of bots with more extreme personality is regarded to be more human. We hypothesise that many participants may not expect machines to display true autonomy; instead, they anticipate the bot to be driven by a simple algorithm, leading to consistent, repetitive behaviour. Among the three bot types, NEUTRAL fits that bill best. What is perhaps missing from our analysis is an attempt to determine whether subjects expect the bot to act human-like. Do they view robots as partners or servants?

Finally, we discuss subjects' self-reported trust changes following game play. The answer distributions from Figure 7.10 are hardly surprising – a majority of subjects lose trust towards non-cooperative GREEDY bot and maintain their trust in other cases. However, given that fairness differs significantly between SELFLESS and NEUTRAL, the fact that there is very little discrepancy between both groups in terms of trust changes is unexpected. It may be attributed to chance or inaccurate reporting, but it could also suggest that gaining human's trust is quite difficult.

A more encouraging conclusion can be drawn from the analysis of the correlation between subjects' reported change of trust towards the bot and their shift in trust towards robots generally, illustrated in Figure 7.11. Note that, if the two answers were perfectly correlated, we would expect the bottom-left to top-right diagonal of the confusion matrix to contain all the values. Even though that is not the case, the correlation is still quite strong, with 67.5% of values falling on the main diagonal. What is interesting, however, are the cases when the two answers do not match. In particular, we note the imbalance in the distribution of these mismatching answers – a vast majority of them corresponds to the case when participant's trust towards the bot decreases, but their trust towards robots remains unchanged (represented by a square marked "22"). In contrast, when participant's trust towards the bot increases, it predominantly translates into an increase in general trust too. Hence, participants seem to display a strong propensity to trust robots; they are willing to generalise positive experiences and tend to treat disappointing outcomes as isolated events.

Minor Results Among the minor results of the study, arguably the most interesting one relates to human opportunism. It should perhaps not come as a surprise that, by and large, participants have chosen to betray the bot's trust to earn a few more dimes. Now, while this suggests that humans do not treat robots as equal partners, we believe caution is due before drawing any conclusions. A crucial aspect of our experimental setting is that it is clear from the onset that this is a one-off interaction. Hence, participants have no incentive to maintain a long-term relationship with the bot. This is a fundamental difference between our experiment and possible real-life human-robot partnerships such as autonomous driving, care of elderly, home assistance or search and rescue.

Still, our finding could prove useful in the design of customer-facing robots that are not owned by a single individual, but provide a service to the general public. It seems such robots could be subject to more abuse than a human would be when performing the same function. Their design should therefore reflect it, as well as try to mitigate it.

As for evaluation of our time control mechanism, the results should be taken with a grain of salt, reflecting the number of assumptions and simplifications we have made. However, at the very least, the method appears promising and further experimentation could bring more definitive answers. It would certainly be interesting to see how questionnaire completion times would differ in the absence of any time controls. In retrospect, we could have included such an analysis in our study, by randomly including the time controls for only half of participants. Unfortunately, some good ideas only come to mind when it is too late.

7.5 Conclusions

We conclude by including selected examples of feedback that subjects left at the end of the study (original spelling).

> "One of it's kind excellent survey, changed my perception towards bot."
>
> "I somewhat didn't expect the bot to learn from my behavior. Even if I was initially greedy, I rather expected the bot to give me a second chance to return some units in a later round."
>
> "The robot seemed inconsistent regarding how much it returned. It was particularly stingy when I invested 4 units, and it returned 0, but seemed to act somewhat fairly for all other investment amounts."
>
> "Its really a nice and unique survey about playing a simple investment game with an artificial agent. Thank you."
>
> "Already don't trust, confirmation of my feelings."
>
> "This bot was a jerk. I couldn't believe they returned 0 units to me on most of the rounds. It makes me feel angry, which is irrational since it is a bot and not a person."
>
> "first 3 rounds i had not given any units so after that bot also didn't give me any units. I think this is called AI. Where they can grasp other person behavior very easily."
>
> "Despite being disappointed by the bot's investments in the last few rounds, I liked playing this game. Thank you :)"

Other comments left by participants can be found in Appendix C. Overall, the feedback was for the most part positive, with many subjects finding the game interesting and stimulating, and theorising about how the bot is implemented.

8 Conclusions and Future Work

Our original goal for this research was formalising trust and providing a way to reason about it. First, we proposed ASMAS, a formal framework that extends partially observable stochastic games with so-called cognitive dimension, which represents mental reasoning of agents. Reasoning about trust is then possible thanks to a specially designed logical language, whose formulas are interpreted on instances of ASMAS.

However, scrutiny of that formalism reveals its inadequacy for modelling human-robot interactions and identifies inconsistencies in proposed trust definitions. We rectify these issues with an updated framework, CSMG; in order to define trust truthfully, we take a step back and consider what motivates humans and how they make decisions. CSMG integrates several existing behavioural models and formulates a novel utility function that captures agents' emotions. In this setting, we put forward a formal definition of trust that exhibits certain theoretically desired properties.

We implement our model in a probabilistic programming language called WebPPL and use the developed tool to analyse a number of case studies. Moreover, we designed and ran a human experiment in which participants play the trust game against a bot run by our model. We have shown that behavioural predictions generated by our framework are no worse than those from a predictor designed specifically for that game, and significantly outperform baseline methods.

We view our work as a proof of concept, showing that there is a place for reasoning about emotions in software deployed to autonomous robots. The most immediate application of our tool is envisaged in home-assistive and other humanoid robots.

They will need to interact with people continuously, for which an understanding of our emotions and motivations will be vital. For that application to materialise, our model will likely need to be ported to another programming language – a task we hope will be facilitated by the fact that the existing implementation is probabilistic. Over a longer horizon, the framework could fuel other forms of automation, such as self-driving cars, allowing detection of too high or too low levels of trust.

Weaknesses Admittedly, some aspects of our work leave space for improvement. For example, the universality of our approach may not be evident due to the relatively small number of case studies presented. This is because of time constraints; we have prioritised other elements of our work, such as the human experiment, which left no time to develop more case studies.

On a related note, we wish we had been able to exhaustively model other mental states the way we expressed trust. As this document shows, a lot of effort went into our formulation of trust and its dynamics model. It is also to be expected that not all emotions will admit an elegant definition like trust did, as we designed our framework with trust in mind. However, we note that not all mental states must be modelled from scratch as we did for trust. Probabilistic affective models should be utilised whenever available; they could serve as the definition of a given emotion, while the dynamics model could be learned from data, in a similar fashion to how Chen et al. [38] do it.

We also remark upon the assumed structure of the utility function. Assigning numerical values to all human motivations, including emotional ones, is disputable. But we believe settling on such a common representation is the only way to compare things like earning \$100 and the guilt experienced for betraying a friend's trust (that might have enabled earning of the \$100). Differing ranges of physical and mental rewards are troubling, but the resulting inconvenience is partly offset by appropriately using the reward utility functions and learning of goal coefficients. It is also important to keep in mind that we do not postulate that all those numbers (utilities) representing various motivations actually arise when average human makes decisions. Instead, the utility function simply reflects the preferences of an agent. Crucially, our model learns the utility function from agents' behaviour, meaning the accuracy of our representation improves as more data becomes available.

A discussion of deficiencies of our work would not be complete without mentioning the suboptimal performance of our tool. This is partly due to the low efficiency of the implementation language, WebPPL. A more modern language optimised for performance could be used instead, Pyro being one candidate. However, the other avenue for improvement is in the algorithms that operate our tool. The main source of complexity is the decision tree traversal that agents perform when computing their actions. Its depth is controlled by the lookahead parameter of an agent, which is uniform among all branches of the tree. However, there are reasons to think that humans are more discriminative in their exploration of the decision tree, discarding some parts of it altogether without deep consideration. Encoding heuristics of this kind could bring a significant improvement to the tool's efficiency and enable us to consider larger lookaheads and wider action ranges for agents.

Future Work Addressing some of the issues mentioned above provides several directions for future work. However, there are a few more avenues for further research which deserve a note. The most obvious one, and perhaps the most difficult too, is to add support for concurrent games. The problem is that sequentiality is ingrained into the decision-making process we have formulated. When agents are allowed to take actions simultaneously, an alternative approach, perhaps based on equilibrium analysis, is needed.

A more conceptually straightforward extension of our framework has to do with the inference mechanism of mental rewards of others. Recall that, currently, agents start with an estimation of their opponents' mental states and update it according to an appropriate dynamics function. However, in reality, another mechanism is at play that allows an agent to improve their estimation – the evidential inference of deducing a value of other's mental state based on their behaviour. This difference is best illustrated using the trust game example. We have assumed that, upon Alice's action, her estimation of Bob's trust is updated and vice versa. However, consider Alice's action of investing (or not). Intuitively, the amount she chooses can be assumed to reflect the amount of trust she places in Bob. Therefore, based on her action, Bob should be able to infer how much she trusts him. This is not the same mechanism as trust estimation update that follows one's own action.

Finally, we remark that our focus in this work has been on the core, dispositional element of trust that most other authors have ignored. However, integrating a competence model would make the framework more robust. Fortunately, this need not require much effort, as we could use any of the models proposed by the robotics community, such as the task-specific trust model due to Soh et al. [39].

Appendices

A
User Guide

Our tool is developed as a WebPPL package, in the same way as standard packages like `webppl-json`[1], `webppl-csv`[2] or a more closely-related `webppl-agents`[3]. It is called `webppl-cognitive-agents`[4].

Prerequisites Being a WebPPL package, `webppl-cognitive-agents` needs a working installation of WebPPL. This, in turn, requires presence of Node.js – a JavaScript runtime. To install both, follow instructions from WebPPL documentation[5].

Installation With WebPPL up and running, `webppl-cognitive-agents` package can be installed in a standard way:

```
> npm install webppl-cognitive-agents
```

Installation can be verified by executing

```
> webppl test/installation.wppl --require .
```

in the top-level directory of the downloaded package.

[1]Available at `https://github.com/stuhlmueller/webppl-json`; last accessed: 01/02/2022

[2]Available at `https://github.com/mhtess/webppl-csv`; last accessed: 01/02/2022

[3]Available at `https://github.com/agentmodels/webppl-agents`; last accessed: 01/02/2022

[4]Available at `https://github.com/maciekolejnik/webppl-cognitive-agents`; last accessed: 01/02/2022

[5]Available at `https://webppl.readthedocs.io/en/master/installation.html`; last accessed: 07/02/2022

Usage To use the package in your code, require it on the command line like this:

```
> webppl <your-code> --require <path to webppl-cognitive-agents>
```

in the same way as you would use any other WebPPL package.

We refer the reader to the `README` file in the top-level directory of the downloaded package for instructions on how to construct models, run simulations and perform inferences.

B
Case Studies Experiments

In this chapter, we provide instructions for replicating the experiments we described in Section 6.2. Note that a working installation of WebPPL along with the `webppl-cognitive-agents` package is required to proceed. Appendix A outlines how to set that up.

B.1 Trust Game

We begin with the trust game experiments. Recall that we tested three hypotheses, H1, H2 and H3. Below we describe how we did it. Note that all the experiments we run are defined inside the `examples/trustgame/src/simulations.wppl` file within the `webppl-cognitive-agents` package.

H1 To verify the first hypothesis, execute

```
> webppl examples/trustgame/src/simulations.wppl --require .
  --require examples/trustgame -- --experiment 0 --scenario 0
```

which runs the first scenario of the first experiment. By default, it produces as output all the action distributions according to which agents select their actions. This can be surpressed by appending `--log 1` to the above command, in which case only the trace is printed.

We obtained the following trace when we run that command: $[4, 5, 4, 4, 4, 4, 4, 4, 4, 4]$. However, $[4, 4, 4, 4, 4, 4, 4, 4, 4, 4]$ is also a likely outcome since Bob selects his first action according to (roughly) the following distribution: $\langle 4 \to 0.25, 5 \to 0.75\rangle$.

Table B.1: Randomly generated parameters of agents in our run of the `varyTrustExperiment`

	Alice		Bob	
	Goal coeffs	Trust estimation	Goal coeffs	Trust estimation
1	[0.99, 0.01]	0.35	[0.17, 0.83]	0.79
2	[0.99, 0.01]	0.92	[0.74, 0.26]	0.43
3	[0.2, 0.8]	0.96	[0.78, 0.22]	0.57
4	[0.74, 0.26]	0.29	[0.94, 0.06]	0.66
5	[0.32, 0.68]	0.15	[0.13, 0.87]	0.57
6	[0.13, 0.87]	0.96	[0.02, 0.98]	0.68
7	[0.19, 0.81]	0.46	[0.65, 0.35]	0.46
8	[0.82, 0.18]	0.45	[0.58, 0.42]	0.52
9	[0.76, 0.24]	0.05	[0.29, 0.71]	0.49
10	[0.41, 0.59]	0.12	[0.87, 0.13]	0.73
11	[0.29, 0.71]	0.35	[0.3, 0.7]	0.86
12	[0.47, 0.53]	0.86	[0.36, 0.64]	0.51
13	[0.72, 0.28]	0.15	[0.22, 0.78]	0.68
14	[0.57, 0.43]	0.54	[0.57, 0.43]	0.2
15	[0.37, 0.63]	0.83	[0.9, 0.1]	0.02
16	[0.35, 0.65]	0.86	[0.9, 0.1]	0.83
17	[0.59, 0.41]	0.13	[0.79, 0.21]	0.6
18	[0.57, 0.43]	0.15	[0.95, 0.05]	0.96
19	[0.09, 0.91]	0.93	[0.27, 0.73]	0.28
20	[0.6, 0.4]	0.8	[0.14, 0.86]	0.6

H2 Verifying the second hypothesis is a more involved process, captured by a custom experiment called `varyTrustExperiment`. To run it, execute

```
> webppl examples/trustgame/src/simulations.wppl --require .
  --require examples/trustgame -- --experiment 1 --log 1 --reps 20
```

This will print all the traces, for each run of each scenario of each experiment. Note that parameters of agents are selected randomly in this experiment, which means replicating the exact results is not trivial. However, Table B.1 lists the parameters that were selected for the run whose results we report in Figure 6.2.

H3 Finally, the third hypothesis concerns conman behaviour. To model this type of agent in the framework, certain special steps have to be taken, which can be consulted by inspecting `conmanCognitiveExperiment` in the simulations file. To run the experiment:

```
> webppl examples/trustgame/src/simulations.wppl --require . --require
    examples/trustgame -- --experiment 3 --log 1
```

By default, it simulates three rounds of the trust game between a standard cognitive agent and a con man twenty times. The number of repetitions may be changed to n by appending `--reps n` to the above command. Note that, due to high lookahead of Bob, the simulations are expected to run for a considerable amount of time (up to an hour).

The trace of each execution is printed to the user and must be inspected manually to interpret the results. The traces will likely vary significantly, reflecting a wide variety of Alice's personalities and beliefs. The experiment calculates average earnings of both players, which are expected to be somewhat higher for Bob.

B.2 Bravery Game

Our analysis of the bravery game consists of two experiments, specified programatically in the file `examples/bravery/src/simulations.wppl` as `equilibriumExperiment` and `varyTrustExperiment`.

The first of these, `equilibriumExperiment`, sets goal coefficients of agents to mimic their utility function given in the original model as a psychological game [52]. Beliefs of players are varied and, for each setting, twenty steps of the bravery game are simulated to see if an equilibrium-like pattern of behaviour arises. To run the experiment, execute

```
> webppl examples/bravery/src/simulations.wppl --require .
  --require examples/bravery -- --experiment 1 --log 1
```

The result will be in the form of three traces (for different values of initial beliefs) – these have to be inspected manually. The paper makes a claim that an equilibrium is reached regardless of initial belief, which will be confirmed if the traces are dominated by *bold* actions from some point onwards.

The second experiment studies how player 1's lookahead affects the total reward accumulated by each player in the game. To run it, execute:

```
> webppl examples/bravery/src/simulations.wppl --require .
  --require examples/bravery -- --experiment 2 --runs 5 --log 1
```

This prints total rewards of players accumulated over 10 rounds for three different values of lookahead (1,3,5), averaged over five runs. The results should match the values in Table 6.2.

B.3 Tipping

Tipping experiments differ from the above in that they involve inference from data. This makes running the experiments a little more cumbersome, despite our best efforts to simplify the process by providing custom scripts.

```
% this file contains a (made up) record of tips given after receiving
% given quality of service
% the tips are medium, characterised by tipping norm of roughly 10-15
% the behaviour is not fully consistent as the agent gives tips of
% various sizes depending on their mood etc

(0,ben);(1,abi)

10;4;4

<start>
% agents
0; 1
% rounds
normal;15
bad;0
good;25
good;20
normal;10
bad;5
bad;10
good;20
normal;20
normal;10
<end>
```

Listing 12: File `mediumTips.csv` from the tipping case study

Inferences from Synthetic Data The first experiment is the easier of the two, as it involves inference from three batches of *synthetic* data, each characterised by low, medium or high tips. The data files are located in the `examples/tipping/data` directory. Listing 12 gives an example of one such file; for the others, we refer the reader to the code base. A script that automates inference from all three batches and reports results to a text file is called `inferFromSyntheticData` and located in the `examples/tipping` directory. It is executed as follows:

```
> bash examples/tipping/inferFromSyntheticData syntheticInfer.txt
```

Note that inference is a computationally intensive process and an execution of that script may take over ten minutes. The results of inference, which consist of three sets (one per data file) of discrete probability distributions (describing posterior over Abi's goal coefficients, her GASP score and the tipping norm) are saved to a file `syntheticInfer.txt` written into the `examples/tipping/results` directory.

Additionally, the priors used for inference are recorded in that file. Recall that separate distributions are used for each parameter we are inferring and they are all uniform, but discretised. Specifically, prior over Abi's goal coefficients is a uniform distribution over the set $\{[a, 1-a] \mid a \in \{1, .8, .6, .4, .2, 0\}\}$, prior over Abi's GASP score has support $\{1, 3, 5, 7\}$ and prior over the tipping norm is uniform over $\{5, 10, 15, 20, 25\}$.

The next experiment that uses synthetic data is designed to illustrate that tipping norm should not be inferred from data alone, as that ignores the confounding effect of agent's goal coefficient and GASP score. To show it, we focus on `mediumTips.csv`, the data file characterised by medium tips, and we study how the inferred posterior differs for various configurations of Abi's goal coefficients and her GASP score. This experiment runs inference ten times; on each occasion, Abi's goal coefficients are set randomly (but each goal coefficient is a multiple of .1, which gives eleven possibilities), as well as her GASP score (seven possible values). To run it, execute

```
> bash examples/tipping/inferFromSyntheticDataFixGoalCoeffsAndGasp
  syntheticInferFixed.txt
```

The results get saved in `syntheticInferFixed.txt`. They must be interpreted by manually inspecting the posterior distributions on the tipping norm in each case. We claim there will be a sizable amount of variability in those posteriors, showing that Abi's parameters must be taken into account when inferring the tipping norm.

Inferences from Generated Data The second type of experiments involves learning from data generated by the tool. This requires the data to be generated first:

```
> bash examples/tipping/generateDataFiles
```

This runs simulations of the type familiar from previous case studies and uses the `webppl-fs` package to construct appropriate data files based on the outcomes of the simulations. They are saved in a subdirectory `generatedData` within the `data` directory; specifically, inside three subdirectories `5`, `10` and `15`, corresponding to the number of rounds of the tipping game that are simulated.

To perform inference based on the generated files, execute the following three commands:

```
> bash examples/tipping/inferFromGeneratedData 5rounds.txt 5 10
> bash examples/tipping/inferFromGeneratedData 10rounds.txt 10 10
> bash examples/tipping/inferFromGeneratedData 15rounds.txt 15 10
```

Each command processes ten files and prints its progress to the user (this is helpful since the commands generally take a long time to run).

To evaluate the posteriors, i.e., generate the MSPEs from Table 6.3, we have included a Python script, which reads the text files (both data files and results files) to obtain observed behaviour and tool's predictions and computes MSPEs of each prediction (separately for goal coefficients, the tipping norm and the GASP score). It then averages the MSPEs, both as means and medians, and prints the results to the user. Execute the following commands to evaluate the predictions:

```
> python3 util/computeMSPE.py 5 10
> python3 util/computeMSPE.py 10 10
> python3 util/computeMSPE.py 15 10
```

The expectation is that the results will match the values in Table 6.3. Of course, due to randomness in the process of data generation, the learning performance may differ between runs, but the trends should remain the same.

C
Trust Game Experiment

Contents

In this chapter, we provide more details about the way our Trust Game experiment was run. In particular, we describe how participants' answers to the pre-game questionnaire are used to generate an appropriate prior (Section C.1), we give full text used to describe the game and the comprehension questions that follow (Section C.2), as well as a full specification of parameters used to instantiate our bot (Section C.3) and all the feedback left by participants (Section C.4). Note that the web application can be accessed online[1] and we encourage the reader to see it for themselves.

C.1 Generating Priors

This section specifies in detail the procedures we employ to generate estimations of participants' cognitive characteristics and beliefs based on the answers they provide as part of the questionnaire.

[1]At `http://trust-game-experiment.herokuapp.com/`; last accessed: 01/02/2022

Lookahead Lookahead of the participant is estimated from the answer to the 11-20 money request game question (see Section 7.1.5). The formal procedure is given in Algorithm 19. In principle, it boils down to conditioning (but only in the INFORMED treatment) the base distribution from line 2 on the provided answer. Note that the distribution is based on the results reported from the original experiment conducted by the authors [122]. Further, note that lookahead values only up to 4 are considered – this reflects performance limitations.

Algorithm 19: estimating participant's lookahead

```
   input  : answer to money request question $answer$, boolean flag
            $informedTreatment$, participant's role $role$
   output : prior over participant's lookahead, expressed as a probability
            distribution
1  function estimateLookahead(answer,informedTreatment,role):
       /* distribution over vs with probs given by ps; based
          on [122]                                              */
2      $d_0 \leftarrow$ Categorical($ps : [1/14, 2/14, 5/14, 5/14, 1/14], vs : [0, 1, 2, 3, 4]$);
3      $est \leftarrow 20 - answer$;
       /* only use the answer in the INFORMED treatment         */
4      if $informedTreatment$ then
           /* operation below uses webppl's factor operator to bias
              the distribution toward values close to est       */
5          $d \leftarrow$ conditionSoft($d_0$, $est$);
6      else
7          $d \leftarrow d_0$;
8      end
       /* if participant plays as investor, simplify estimation
          for performance reasons                               */
9      if $role$ is investor then
10         $v \leftarrow$ to_integer(expectation($d$));
11         return Delta(v);               /* Dirac centered on v */
12     end
13     return d;
14 end
```

The soft conditioning from line 5 is achieved by using the `factor` operator of WebPPL, as outlined in Algorithm 20. The pseudocode contained therein very closely resembles valid WebPPL code and illustrates a basic but common pattern of inference. It achieves the task of modifying probabilities assigned to values in the support of a given discrete distribution so that elements that are near a given value v are deemed more likely.

```
Algorithm 20: softly conditioning a distribution on an observation
input  : (discrete) probability distribution prior, (observed) value v
         (usually from the support of prior but not necessarily)
output : posterior distribution obtained by softly conditioning prior on
         observing v
1 function conditionSoft(prior,v):
2     function model()
3         s ← sample(prior);
          /* factor(x) adds x to the log probability of current
             execution                                          */
4         factor(|s − v|);  /* the closer s is to v, the more likely */
5         return s;
6     end
      /* Infer performs inference; in this case, by enumerating
         possible executions of model                           */
7     return Infer(method: 'enumerate', model);
8 end
```

With that settled, it remains to explain the if statement of line 9. It simplifies the resulting distribution in the case when the participant has been assigned the role of investor. This is because, as mentioned in Section 7.2.3, when bot is an investee, its lookahead is set to 4. This places a high burden on the computational resources, meaning that mental state and meta-parameter estimations must be kept simple. Given that the time needed to compute bot's action is linear in the product of sizes of supports of the distributions, they must be kept to minimum, hence Dirac delta is used.

Rationality Recall that rationality of a participant is estimated based on the preferences between lotteries that we elicit as part of the pre-game questionnaire. Each user is presented with the following three questions, designed to be progressively more difficult:

1. Imagine you are offered two lotteries defined as follows:

   (A) you receive \$100 with probability 0.6 and \$50 with probability 0.4

   (B) you receive \$100 with probability 0.4 and \$50 with probability 0.6

2. This time, consider the following lotteries:

   (A) you receive \$15 with probability 0.2, \$10 with probability 0.2 and \$5 with probability 0.6

   (B) you receive \$10 with probability 0.8 and \$5 with probability 0.2

3. Finally, consider two lotteries:

   (A) you receive \$100 with probability 0.2, \$50 with probability 0.3 and \$10 with probability 0.5

   (B) you receive \$40 with certainty

In each case, participant is asked to specify whether, given choice, they would pick lottery A, lottery B or if they are indifferent. We expect subjects's preferences to be consistent with expected utility maximisation principle; in particular, they should favour option A in the first question, option B in the second one and be indifferent about the last choice. Note that we have carefully designed the lotteries to reduce the effect various confounders, such as risk aversion or certainty effect [98] that most humans display. Specifically, lotteries do not involve any losses, the amounts are moderate (which ensures utility is approximately linear) and outcomes are predominantly uncertain. We note however that the last question sets a probabilistic lottery against a deterministic one, meaning risk aversion may influence subject's answer. Although our framework does not currently model risk aversion of agents, a future extension might do, in which case our data could be used to estimate that quality.

The formal procedure is given as Algorithm 21. The key observation is that since the distribution of rationality in a population is assumed to follow the bell curve, we would like to represent its estimation with a distribution that is close to being normal. However, for performance reasons, we are limited to discrete distributions with fairly small number of values in its support. Therefore, Algorithm 21 returns a categorical distribution with one or three elements in its support (depending on role assignment in the game, cf. lookahead assignment above) that are sampled from a normal distribution with appropriately chosen mean and standard deviation. That computation is captured by the `normalSamples` function. Mean is set by an iterative process that repeatedly increments or decrements the base value of 16 depending on the correctness of each answer, where easier questions produce bigger modifications. Standard deviation is either 8 or 16 depending on whether answers are consistent or not.

Algorithm 21: estimating participant's rationality

```
input  : lottery preferences answers, boolean flag informedTreatment,
         participant's role role
output : prior over participant's rationality, expressed as a probability
         distribution
function estimateRationality(answers,informedTreatment,role):
    $\mu \leftarrow 16$;                                  /* base rationality */
    $\delta \leftarrow 16$;
    for $i \leftarrow 1$ to 3 do
        if $i^{th}$ preference in answers is correct then
            $\mu \leftarrow \mu + \delta$;
        else
            $\mu \leftarrow \max(0, \mu - \delta)$;
        end
        $\delta \leftarrow \delta/2$;
    end
    /* ternary operator below; standard deviation is lower if
       answers consistent (all wrong or all correct)            */
    $\sigma \leftarrow$ all answers correct or all incorrect ? 8 : 16;
    $s \leftarrow$ role is investor ? 1 : 3;
    if informedTreatment then
        return normalSamples($\mu$, $\sigma$, $s$);
    else
        return normalSamples(32, 8, $s$);
    end
end
```

Belief Finally, we analyse how belief of the bot, represented by a vector $\vec{\alpha} = \langle \alpha_1, \alpha_2 \rangle$ is generated. Recall that belief in this case is virtually equivalent to trust; hence the question we ask participants to estimate it attempts to gauge their trustworthiness. In particular, we ask "On a scale from 1 to 5, how concerned are you with the way the robot perceives you?" and hypothesise that the bigger the answer, the more trustworthy (towards the bot) the subject is. Therefore, we designate five different beliefs, one for each possible answer, that are characterised by varying values of trust. The exact formula is $\vec{\alpha} = \langle 2.25 - 0.25a, 0.75 + 0.25a \rangle$, where $a \in \{1, 2, 3, 4, 5\}$ is the answer provided. This gives fairly neutral belief values, ranging from $[2, 1]$ to $[1, 2]$, which means initial trust ranges from 2/3 to 1/3. Note that the sum of entries of $\vec{\alpha}$ roughly corresponds to the confidence in the accuracy of the belief – in this case, it is quite low due to unreliability of human self-reporting. In fact, this confidence is even lower in the UNIFORM treatment, where belief is set to $\vec{\alpha} = [1, 1]$, reflecting lack of knowledge.

C.2 Game Description

For completeness, we include the full description of the game that was presented to participants of our experiment. We begin by introducing the game:

> You will now play a simple game called Trust Game against a *bot*. We explain what that means below. Please read the description carefully as we will ask you questions to ensure understanding. You will get two chances to answer the questions correctly; if you don't, your submission will be rejected.

That is followed by an overview of the payment structure, which allows us to introduce the abstract "units" that we use as currency in the experiment. The text below in under a headline that says "Currency".

> The Trust Game is a money-exchange game. However, instead of dollars, we use abstract monetary *units*, where 20 units = \$1, 4 units = \$0.20, 1 unit = \$0.05 etc.

The game description (headlined "Game Description") then follows; it starts by stating the game horizon, or lack thereof. In the DISCLOSED treatment, it reads:

> The game is played through seven identical rounds,

while in the UNDISCLOSED treatment, an explanation is given:

> The game is played through a number of identical rounds. Because of the experimental condition you have been assigned to, we can't tell you exactly how many rounds will be played, but it will be more than 2 and no more than 20.

Next, mechanics of the game are described; note that the wording again differs, this time depending on the role assigned to the participant. If they are an investor, they are shown the following:

> In **each round** you receive 4 units, and you can **invest** any part of your endowment by giving it to the robot. The amount you invest gets **doubled** in transit. Subsequently the robot can **return** to you any part of what it has received. For example, if out of your 4 units you invest 3 with the robot, it will receive 6, and then it may return to you anything between 0 and 6.

Otherwise, they see this:

> In **each round** the bot receives 4 units, and it can **invest** any part of its endowment by giving it to you. The amount the bot invests gets **doubled** in transit. Subsequently you can **return** to the robot any part of what you have received. For example, if out of its 4 units the bot invests 3 with you, you will receive 6, and then you may return to the bot anything between 0 and 6.

Finally, we include a clarification on how earnings are accumulated throughout the game. For investors:

> Your earnings in the game get accumulated from round to round, but in each single round you can still invest only up to 4 units.

While for investees:

> Your earnings in the game get accumulated from round to round, but in each single round you can return to the bot only up to as many units as you received in that round. Similarly, in each round, the bot can invest only up to 4 units.

Comprehension Check Presentation of the game description is followed by a comprehension check, consisting of three questions designed to exhaustively verify participant's understanding of the Trust Game. The exact wording of the questions varies depending on subject's role; below we consider the case when they are an investor.

1. Suppose you invested 3 units. How many units will the bot receive?
2. Suppose the bot received 8 units. What's the maximum number of units they can share?
3. Suppose you earned 3 units in the first round. What amount will you have available for investment in the second round?

Each question is answered by selecting a value from a drop-down list. In particular, possible responses are: $\{0, 3, 6, 9\}$, $\{0, 4, 6, 8\}$, $\{0, 3, 4, 7\}$ for questions one, two and three, respectively. Subjects have two opportunities to find the right answers; in case not all responses are correct in their first attempt, participants are told they have their last chance, but we do not tell them which answers were wrong.

C.3 Bot Setup

Even though we have already talked about the parameterisation of the bot in Section 7.1.3, we formally summarise our choices below. The goal coefficients of the bot depend on its randomly selected character, given by the mapping: $\langle \text{selfless} \rightarrow [.3, .7], \text{neutral} \rightarrow [.5, .5], \text{greedy} \rightarrow [.8, .2] \rangle$. Further, meta-parameters of the bot are set as follows: rationality $\alpha_{\text{bot}} = 1000$ reflects the infallibility of machine reasoning; discount factor $\gamma_{\text{bot}} = 0.8$ models the uncertainty about game's horizon (recall that the bot does not know how many rounds the game will run for, regardless of the treatment); lookahead $\beta_{\text{bot}} \in \{2, 4\}$ differs depending on the role assignment – it is higher when bot is an investee.

C.4 Participants' Comments

We include all the feedback that participants voluntarily left after completing the experiment. It is presented in Table C.1, along with information specifying the treatment each subject was assigned to.

Table C.1: Comments left by participants (original spelling) with additional information about treatment they were assigned to ("role" refers to the subject while "character" is that of the bot)

Role	Horizon	Character	Feedback
INVESTOR	DISCLOSED	NEUTRAL	This was interesting. I liked trying to figure out how the bot would choose how much to send back.
INVESTOR	DISCLOSED	GREEDY	I know the bot is just programmed to behave a certain way so it's just about what the human that programmed it wants
INVESTEE	DISCLOSED	SELFLESS	first 3 rounds i had not given any units so after that bot also didn't give me any units. I think this is called AI. Where they can grasp other person behavior very easily.
INVESTOR	DISCLOSED	SELFLESS	Great game.
INVESTEE	DISCLOSED	SELFLESS	It was interesting Game, play with bot. Thank you!
INVESTOR	DISCLOSED	GREEDY	This bot was a jerk. I couldn't believe they returned 0 units to me on most of the rounds. It makes me feel angry, which is irrational since it is a bot and not a person.

Continued on next page

Table C.1 – continued from previous page

Role	Horizon	Character	Feedback
INVESTEE	DISCLOSED	NEUTRAL	The robot got stingy towards the later rounds.
INVESTOR	DISCLOSED	NEUTRAL	I thought the game was interesting. I wasn't sure how the bot would react to my decisions. The descriptions were pretty clear to me. I had no problems.
INVESTOR	UNDISCLOSED	NEUTRAL	I trust the robot
INVESTOR	DISCLOSED	GREEDY	Already don't trust, confirmation of my feelings.
INVESTEE	UNDISCLOSED	GREEDY	Its really a nice and unique survey about playing a simple investment game with an artificial agent. Thank you.
INVESTOR	UNDISCLOSED	GREEDY	No issues, questions, comments, or concerns, thanks!
INVESTOR	UNDISCLOSED	SELFLESS	Game broke at first, refresh fixed it.
INVESTOR	UNDISCLOSED	GREEDY	Seems like the Bot has been programmed with lower odds for returning coins.
INVESTOR	UNDISCLOSED	NEUTRAL	The robot seemed inconsistent regarding how much it returned. It was particularly stingy when I invested 4 units, and it returned 0, but seemed to act somewhat fairly for all other investment amounts.
INVESTOR	DISCLOSED	NEUTRAL	I enjoyed the game. Thank you.
INVESTOR	DISCLOSED	NEUTRAL	directions were easy to understand and I appreciate they were present when the comprehension questions were asked
INVESTOR	DISCLOSED	GREEDY	just seems like the robot was going to keep all every time regardless
INVESTOR	UNDISCLOSED	GREEDY	wished i could play more rounds and play with a different robot!
INVESTEE	DISCLOSED	NEUTRAL	I somewhat didn't expect the bot to learn from my behavior. Even if I was initially greedy, I rather expected the bot to give me a second chance to return some units in a later round.
INVESTOR	UNDISCLOSED	NEUTRAL	I had fun playing the game and the instructions were clear. Thank you.
INVESTOR	UNDISCLOSED	GREEDY	One of it's kind excellent survey, changed my perception towards bot.
INVESTOR	DISCLOSED	GREEDY	Nice Survey, having opportunity to earn bonus!
INVESTEE	DISCLOSED	NEUTRAL	I found it interesting.

Continued on next page

Table C.1 – continued from previous page

Role	**Horizon**	**Character**	**Feedback**
INVESTOR	DISCLOSED	SELFLESS	this was fun, thank you
INVESTEE	UNDISCLOSED	SELFLESS	Despite being disappointed by the bot's investments in the last few rounds, I liked playing this game. Thank you :)
INVESTEE	UNDISCLOSED	NEUTRAL	Interesting little game; I have a small background in programming, so I figured that eventually the bot would reduce the investment as I reduced my return. Thanks, and good luck!
INVESTOR	UNDISCLOSED	SELFLESS	I was hoping the bot would send me more since I sent everything but still, not bad.

References

[1] Georg Graetz and Guy Michaels. “Robots at Work”. In: *The Review of Economics and Statistics* 100.5 (2018), pp. 753–768.

[2] J. D. Lee and K. A. See. “Trust in Automation: Designing for Appropriate Reliance”. In: *Human Factors: The Journal of the Human Factors and Ergonomics Society* 46.1 (2004), pp. 50–80.

[3] Laura Fiorini et al. “Assistive robots to improve the independent living of older persons: results from a needs study”. In: *Disability and Rehabilitation: Assistive Technology* 16.1 (2021), pp. 92–102.

[4] Alkis Papadoulis, Mohammed Quddus, and Marianna Imprialou. “Evaluating the safety impact of connected and autonomous vehicles on motorways”. In: *Accident Analysis and Prevention* 124 (2019), pp. 12–22.

[5] Philipp Tschandl et al. “Human–computer collaboration for skin cancer recognition”. In: *Nature Medicine* 26.8 (2020), pp. 1229–1234.

[6] Ross D. Arnold, Hiroyuki Yamaguchi, and Toshiyuki Tanaka. “Search and rescue with autonomous flying robots through behavior-based cooperative intelligence”. In: *Journal of International Humanitarian Action* 3.1 (2018).

[7] Stuart J. Russell. *Human compatible: Artificial intelligence and the problem of control.* Penguin, 2019.

[8] David Leslie. “Raging robots, hapless humans: the AI dystopia”. In: *Nature* 574.7776 (2019), pp. 32–33.

[9] Dave Lee. *US opens investigation into Tesla after fatal crash.* Ed. by British Broadcasting Corporation (BBC) News. [Online; posted 1-July-2016; last accessed 5-November-2021]. 2016. URL: `https://www.bbc.co.uk/news/technology-36680043`.

[10] David Shepardson. *Tesla Autopilot engaged in 2018 California crash; driver’s hands off wheel: NTSB.* Ed. by Reuters. [Online; posted 3-September-2019; last accessed 5-November-2021]. 2019. URL: `https://www.reuters.com/article/us-tesla-crash-idUSKCN1VO22E`.

[11] Bryan Pietsch. *2 Killed in Driverless Tesla Car Crash, Officials Say.* Ed. by The New York Times. [Online; posted 18-April-2021; last accessed 5-November-2021]. 2021. URL: `https://www.nytimes.com/2021/04/18/business/tesla-fatal-crash-texas.html`.

[12] Alan R. Wagner, Jason Borenstein, and Ayanna Howard. “Overtrust in the robotic age”. In: *Communications of the ACM* 61.9 (2018), pp. 22–24.

[13] Victoria A. Banks, Katherine L. Plant, and Neville A. Stanton. “Driver error or designer error: Using the Perceptual Cycle Model to explore the circumstances surrounding the fatal Tesla crash on 7th May 2016”. In: *Safety Science* 108 (2018), pp. 278–285.

[14] Shili Sheng et al. “Trust-based route planning for automated vehicles”. In: *Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems*. New York, NY, USA: ACM, 2021, pp. 1–10.

[15] Peter Apps. *New era of robot war may be underway unnoticed*. Ed. by Reuters. [Online; posted 10-June-2021; last accessed 5-November-2021]. 2021. URL: `https://www.reuters.com/article/apps-drones-idUSL5N2NS2E8`.

[16] United Nations Panel of Experts on Libya. *Report*. 2021. URL: `https://digitallibrary.un.org/record/3905159` (visited on 11/04/2021).

[17] Benjamin Kuipers. “How can we trust a robot?” In: *Communications of the ACM* 61.3 (2018), pp. 86–95.

[18] Lola Cañamero. “Emotion understanding from the perspective of autonomous robots research”. In: *Neural Networks* 18.4 (2005), pp. 445–455.

[19] Cynthia Breazeal. “Role of expressive behaviour for robots that learn from people”. In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 364.1535 (2009), pp. 3527–3538.

[20] Morteza Lahijanian and Marta Kwiatkowska. “Social Trust : a Major Challenge for the Future of Autonomous Systems”. In: *AAAI Fall Symposium on Cross-Disciplinary Challenges for Autonomous Systems*. AAAI Press, 2016, pp. 189–193.

[21] Olivia Nocentini et al. “A survey of behavioral models for social robots”. In: *Robotics* 8.3 (2019).

[22] R.W. Picard. *Affective computing*. 2000.

[23] Desmond Ong et al. “Applying Probabilistic Programming to Affective Computing”. In: *IEEE Transactions on Affective Computing* 12.2 (2019), pp. 306–317.

[24] Simon Baron-Cohen, Alan M. Leslie, and Uta Frith. “Does the autistic child have a ’theory of mind’ ?” In: *Cognition* 21.1 (1985), pp. 37–46.

[25] Yann Algan and Pierre Cahuc. “Inherited trust and growth”. In: *American Economic Review* 100.5 (2010), pp. 2060–2092.

[26] Paul Ward and Samantha Meyer. “Trust, Social Quality and Wellbeing : A Sociological Exegesis”. In: *Development and Society* 38.2 (2009), pp. 339–363.

[27] Robert D Putnam. “Social Capital and Public Affairs”. In: *Bulletin of the American Academy of Arts and Sciences* 47.8 (1994), p. 5.

[28] Bing Cai Kok and Harold Soh. “Trust in Robots: Challenges and Opportunities”. In: *Current Robotics Reports* 1.4 (2020), pp. 297–309.

[29] Xiaowei Huang, Marta Kwiatkowska, and Maciej Olejnik. “Reasoning about Cognitive Trust in Stochastic Multiagent Systems”. In: *ACM Transactions on Computational Logic* 20.4 (2019), pp. 1–64.

[30] Joyce Berg, John Dickhaut, and Kevin McCabe. “Trust, Reciprocity, and Social History”. In: *Games and Economic Behavior* 10.1 (1995), pp. 122–142.

[31] Yao Wang and Julita Vassileva. “Bayesian network-based trust model”. In: *Proceedings IEEE/WIC International Conference on Web Intelligence.* IEEE Comput. Soc, 2003, pp. 372–378.

[32] Tina Setter, Andrea Gasparri, and Magnus Egerstedt. “Trust-based interactions in teams of mobile agents”. In: *2016 American Control Conference (ACC).* IEEE, 2016, pp. 6158–6163.

[33] Serena Villata et al. “A socio-cognitive model of trust using argumentation theory”. In: *International Journal of Approximate Reasoning* 54.4 (2013), pp. 541–559.

[34] John Lee and Neville Moray. “Trust, control strategies and allocation of function in human-machine systems”. In: *Ergonomics* 35.10 (1992), pp. 1243–1270.

[35] Anqi Xu and Gregory Dudek. “OPTIMo : Online Probabilistic Trust Inference Model for Asymmetric Human-Robot Collaborations”. In: *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction* (2015), pp. 221–228.

[36] Michael W. Floyd, Michael Drinkwater, and David W. Aha. “Trust-guided behavior adaptation using case-based reasoning”. In: *IJCAI International Joint Conference on Artificial Intelligence* (2015), pp. 4261–4267.

[37] Kristin E. Schaefer et al. “A Meta-Analysis of Factors Influencing the Development of Trust in Automation”. In: *Human Factors: The Journal of the Human Factors and Ergonomics Society* 58.3 (2016), pp. 377–400.

[38] Min Chen et al. “Trust-Aware Decision Making for Human-Robot Collaboration”. In: *ACM Transactions on Human-Robot Interaction* 9.2 (2020), pp. 1–23.

[39] Harold Soh et al. “The Transfer of Human Trust in Robot Capabilities across Tasks”. In: *Robotics: Science and Systems XIV.* New York, NY, USA: Robotics: Science and Systems Foundation, 2018, pp. 241–242.

[40] Alan R. Wagner, Paul Robinette, and Ayanna Howard. “Modeling the human-robot trust phenomenon: A conceptual framework based on risk”. In: *ACM Transactions on Interactive Intelligent Systems* 8.4 (2018).

[41] Michael E. Bratman. *Intention, Plans, and Practical Reason.* Cambridge, MA: Harvard University Press, 1987.

[42] B Van Linder, W van der Hoek, and J Meyer. “Actions that make you change your mind”. In: *KI-95: Advances in Artificial Intelligence* (1995), pp. 185–196.

[43] Philip R. Cohen and Hector J. Levesque. “Intention is choice with commitment”. In: *Artificial Intelligence* 42.2-3 (1990), pp. 213–261.

[44] A.S. Rao and M.P. Georgeff. “Modeling rational agents within a BDI-architecture”. In: *Readings in agents* (1997), pp. 317–328.

[45] Cristiano Castelfranchi and Rino Falcone. “Social Trust : A Cognitive Approach”. In: *Trust and deception in virtual societies* (2001).

[46] Andreas Herzig et al. “A logic of trust and reputation”. In: *Logic Journal of the IGPL* 18.1 (2009), pp. 214–244.

[47] Ward Edwards. “Subjective Probabilities Inferred From Decisions”. In: *Psychological Review* 69.2 (1962), pp. 109–135.

[48] W Fellner. *Probability and Profit: A Study of Economic Behavior Along Bayesian Lines*. Irwin series in economics. R.D. Irwin, 1965.

[49] Bengt Hansson. “The Appropriateness of the Expected Utility Model”. In: *Erkenntnis* 9.2 (1975), pp. 175–193.

[50] Ido Erev et al. “From anomalies to forecasts: Toward a descriptive model of decisions under risk, under ambiguity, and from experience”. In: *Psychological Review* 124.4 (2017), pp. 369–409.

[51] Ori Plonsky et al. “Psychological forest: Predicting human behavior”. In: *31st AAAI Conference on Artificial Intelligence* (2017), pp. 656–662.

[52] John Geanakopolos, David Pearce, and Ennio Stacchetti. “Psychological games and sequential reciprocity”. In: *Games and Economic Behavior* 1 (1989), pp. 60–79.

[53] Pierpaolo Battigalli and Martin Dufwenberg. “Dynamic psychological games”. In: *Journal of Economic Theory* 144.1 (2009), pp. 1–35.

[54] Noel D. Johnson and Alexandra A. Mislin. “Trust games: A meta-analysis”. In: *Journal of Economic Psychology* 32.5 (2011), pp. 865–889.

[55] Atte Oksanen et al. “Trust Toward Robots and Artificial Intelligence: An Experimental Approach to Human–Technology Interactions Online”. In: *Frontiers in Psychology* 11 (2020), pp. 1–13.

[56] E. Schniter, T.W. Shields, and D. Sznycer. “Trust in humans and robots: Economically similar but emotionally different”. In: *Journal of Economic Psychology* 78 (2020), p. 102253.

[57] Roberta C Ramos Mota et al. “Playing the ‘trust game’ with robots: Social strategies and experiences”. In: *2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*. IEEE, 2016, pp. 519–524.

[58] Julian B. Rotter. “A new scale for the measurement of interpersonal trust”. In: *Journal of Personality* 35.4 (1967), pp. 651–665.

[59] Diego Gambetta. *Trust: Making and Breaking Cooperative Relations*. Oxford: Basil Blackwell, 1988, p. 246.

[60] Roger C. Mayer, James H Davis, and F. David Schoorman. “An Integrative Model of Organizational Trust”. In: *The Academy of Management Review* 20.3 (1995), p. 709.

[61] Morris Rosenberg. *Occupation and Values*. Glencoe, IL: Free House, 1957.

[62] J.K. Rempel, J.G. Holmes, and M.P. Zanna. “Trust in close relationships”. In: *Journal of Personality and Social Psychology* 49.1 (1985), pp. 95–112.

[63] Bonnie M. Muir. “Trust in automation: Part I. Theoretical issues in the study of trust and human intervention in automated systems”. In: *Ergonomics* 37.11 (1994), pp. 1905–1922.

[64] S. Kauppinen, C. Brain, and M. Moore. “European medium-term conflict detection field trials [ATC]”. In: *Proceedings. The 21st Digital Avionics Systems Conference* (2002), pp. 2C1–2C1.

[65] Rosemarie E. Yagoda and Douglas J. Gillan. “You Want Me to Trust a ROBOT? The Development of a Human-Robot Interaction Trust Scale”. In: *International Journal of Social Robotics* 4.3 (2012), pp. 235–248.

[66] Paul Robinette and Ayanna M. Howard. “Trust in emergency evacuation robots”. In: *2012 IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2012* (2012).

[67] Paul Robinette, Ayanna M. Howard, and Alan R. Wagner. “Timing Is Key for Robot Trust Repair”. In: *Lecture Notes in Computer Science.* Lecture Notes in Computer Science 9388 (2015). Ed. by Adriana Tapus et al., pp. 574–583.

[68] Edward L. Glaeser et al. “Measuring trust”. In: *Quarterly Journal of Economics* 115.3 (2000), pp. 811–846.

[69] Alexander Todorov. “Evaluating faces on trustworthiness: An extension of systems for recognition of emotions signaling approach/avoidance behaviors”. In: *Annals of the New York Academy of Sciences* 1124 (2008), pp. 208–224.

[70] J. S. Winston et al. “Automatic and intentional brain responses during evaluation of trustworthiness of faces”. In: *Social Neuroscience: Key Readings* (2013), pp. 199–210.

[71] Brian W. Haas et al. “The tendency to trust is reflected in human brain structure”. In: *NeuroImage* 107 (2015), pp. 175–181.

[72] F. Krueger et al. “Neural correlates of trust”. In: *Proceedings of the National Academy of Sciences* 104.50 (2007), pp. 20084–20089.

[73] Angelika Dimoka. “What does the brain tell us about trust and distrust? Evidence from a functional neuroimaging study”. In: *MIS Quarterly* 34.2 (2010), pp. 373–396.

[74] Munjal Desai et al. “Impact of robot failures and feedback on real-time trust”. In: *ACM/IEEE International Conference on Human-Robot Interaction* (2013), pp. 251–258.

[75] Paul Robinette et al. “Overtrust of robots in emergency evacuation scenarios”. In: *11th ACM/IEEE International Conference on Human-Robot Interaction.* 2016, pp. 101–108.

[76] Shili Sheng et al. “A Case Study of Trust on Autonomous Driving”. In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC).* IEEE, 2019, pp. 4368–4373.

[77] P Billingsley. *Probability and Measure.* Wiley Series in Probability and Statistics. Wiley, 1995.

[78] Eli Bingham et al. “Pyro: Deep Universal Probabilistic Programming”. In: *Journal of Machine Learning Research* (2018).

[79] Noah D Goodman and Andreas Stuhlmüller. *The Design and Implementation of Probabilistic Programming Languages.* `http://dippl.org`. [Online; last accessed 11-February-2022]. 2014.

[80] David Tolpin et al. "Design and Implementation of Probabilistic Programming Language Anglican". In: *arXiv preprint arXiv:1608.05263* (2016).

[81] John G. Kemeny and J. Laurie Snell. *Finite Markov Chains*. New York, NY: Springer, 1976, p. 226.

[82] Vojtěch Forejt et al. "Automated verification techniques for probabilistic systems". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6659 LNCS (2011), pp. 53–113.

[83] Christel Baier and Joost-Pieter Katoen. *Principles Of Model Checking*. Vol. 950. 2008, pp. I–XVII, 1–975.

[84] Oskar Morgenstern John Von Neumann. *Theory of Games and Economic Behavior*. Princeton, NJ: Princeton University Press, 1944.

[85] John Nash. "Non-Cooperative Games". In: *The Annals of Mathematics* 54.2 (1951), p. 286.

[86] J. F. Nash. "Equilibrium points in n-person games". In: *Proceedings of the National Academy of Sciences* 36.1 (1950), pp. 48–49.

[87] Reinhard Selten. "Spieltheoretische Behandlung eines Oligopolmodells mit Nachfrageträgheit". In: *Zeitschrift Für Die Gesamte Staatswissenschaft* 121 (1965), pp. 301–324.

[88] David M. Kreps and Robert Wilson. "Sequential Equilibria". In: *Econometrica* 50.4 (1982), p. 863.

[89] R. Selten. "Reexamination of the perfectness concept for equilibrium points in extensive games". In: *International Journal of Game Theory* 4.1 (1975), pp. 25–55.

[90] John C. Harsanyi. "Games with incomplete information played by 'Bayesian' players, I-III: Part I. The Basic Model". In: *Management Science* 50.12 (1968), pp. 1804–1824.

[91] John C. Harsanyi. "Games with Incomplete Information Played by 'Bayesian' Players , I-III . Part II . Bayesian Equilibrium Points". In: *Management Science* 14.5 (1968), pp. 320–334.

[92] John C. Harsanyi. "Games with Incomplete Information Played by 'Bayesian' Players , I-III . Part III . The Basic Probability Distribution of the Game". In: *Management Science* 14.7 (1968), pp. 486–502.

[93] Ward Edwards. "The theory of decision making". In: *Psychological Bulletin* (1954).

[94] Leonard J Savage. *The Foundation of Statistics*. New York, NY, USA: Wiley, 1954, p. 294.

[95] Ariel Rosenfeld and Sarit Kraus. "Predicting Human Decision-Making: From Prediction to Action". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 12.1 (2018), pp. 1–150.

[96] Richard D. McKelvey and Thomas R. Palfrey. *Quantal response equilibria for normal form games*. 1995.

[97] Colin F. Camerer, Teck Hua Ho, and Juin Kuan Chong. "A cognitive hierarchy model of games". In: *Quarterly Journal of Economics* 119.3 (2004), pp. 861–898.

[98] Daniel Kahneman and Amos Tversky. “Prospect Theory: An Analysis of Decision under Risk”. In: *Econometrica* 47.2 (1979), p. 263.

[99] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529.7587 (2016), pp. 484–489.

[100] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. “Anticipating Visual Representations from Unlabeled Video”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* Las Vegas, NV, USA: IEEE, 2016, pp. 98–106.

[101] Mariusz Bojarski et al. “End to End Learning for Self-Driving Cars”. 2016. URL: `http://arxiv.org/abs/1604.07316`.

[102] Gali Noti et al. “Behavior-Based Machine-Learning: A Hybrid Approach for Predicting Human Decision Making”. 2016. URL: `http://arxiv.org/abs/1611.10228`.

[103] Avi Rosenfeld et al. “NegoChat: A chat-based negotiation agent”. In: *13th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2014* 1 (2014), pp. 525–532.

[104] Ariel Rosenfeld and Sarit Kraus. “Providing arguments in discussions on the basis of the prediction of human argumentative behavior”. In: *ACM Transactions on Interactive Intelligent Systems* 6.4 (2016), pp. 1–34.

[105] Benjamin Kuipers. “What is Trust and How Can My Robot Get Some?” In: *a Talk at Social Trust in Autonomous Robots, a workshop in Robotics: Science and Systems 2016*. [Online; last accessed 11-February-2022]. 2016. URL: `http://qav.comlab.ox.ac.uk/trust_in_autonomy/img/KuipersTrustWorkshop16.pdf`.

[106] Omid Madani, Steve Hanks, and Anne Condon. “On the undecidability of probabilistic planning and related stochastic optimization problems”. In: *Artificial Intelligence* 147.1-2 (2003), pp. 5–34.

[107] Daniel Kahneman. *Thinking, fast and slow.* New York, NY, US: Farrar, Straus and Giroux, 2011, p. 499.

[108] Jacob K. Goeree and Charles A. Holt. “Ten little treasures of game theory and ten intuitive contradictions”. In: *American Economic Review* 91.5 (2001), pp. 1402–1422.

[109] Ori Plonsky et al. “Predicting human decisions with behavioral theories and machine learning”. 2019. URL: `http://arxiv.org/abs/1904.06866`.

[110] Dale O. Stahl and Paul W. Wilson. “Experimental evidence on players’ models of other players”. In: *Journal of Economic Behavior and Organization* 25.3 (1994), pp. 309–327.

[111] Elizabeth Bruch and Fred Feinberg. “Decision-Making Processes in Social Contexts”. In: *Annual Review of Sociology* 43 (2017), pp. 207–227.

[112] Amos Tversky and Daniel Kahneman. “Judgment under uncertainty: Heuristics and biases”. In: *Judgment under uncertainty* 185 (2013), pp. 3–20.

[113] Russell Hardin. *Trust and trustworthiness.* Russell Sage Foundation, 2002.

[114] Gene M. Alarcon et al. “The effect of propensity to trust and perceptions of trustworthiness on trust behaviors in dyads”. In: *Behavior Research Methods* 50.5 (2018), pp. 1906–1920.

[115] Owain Evans et al. *Modeling Agents with Probabilistic Programs.* `http://agentmodels.org`. [Online; last accessed 11-February-2022]. 2017.

[116] Noah D Goodman, Joshua B. Tenenbaum, and The ProbMods Contributors. *Probabilistic Models of Cognition.* `http://probmods.org/v2`. [Online; last accessed 11-February-2022]. 2016.

[117] Taya R. Cohen et al. “Introducing the GASP Scale: A New Measure of Guilt and Shame Proneness”. In: *Journal of Personality and Social Psychology* 100.5 (2011), pp. 947–966.

[118] Carlos Alós-Ferrer and Federica Farolfi. “Trust Games and Beyond”. In: *Frontiers in Neuroscience* 13 (2019), pp. 1–14.

[119] James C. Cox. “How to identify trust and reciprocity”. In: *Games and Economic Behavior* 46.2 (2004), pp. 260–281.

[120] Ananish Chaudhuri and Lata Gangadharan. “An Experimental Analysis of Trust and Trustworthiness”. In: *Southern Economic Journal* 73.4 (2007), pp. 959–985.

[121] Quang Vinh Dang and Claudia Lavinia Ignat. “Computational trust model for repeated trust games”. In: *15th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE, 2016, pp. 34–41.

[122] Ayala Arad and Ariel Rubinstein. “The 11–20 Money Request Game: A Level- k Reasoning Study”. In: *American Economic Review* 102.7 (2012), pp. 3561–3573.

[123] Serena Booth et al. “Piggybacking Robots”. In: *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction.* New York, NY, USA: ACM, 2017, pp. 426–434.