Using GDL to Represent Domain Knowledge for Automated Negotiations

Dave de Jonge
Western Sydney University
Sydney, New South Wales, Australia
d.dejonge@westernsydney.edu.au

Dongmo Zhang
Western Sydney University
Sydney, New South Wales, Australia
d.zhang@westernsydney.edu.au

ABSTRACT

Current negotiation algorithms often assume that utility has an explicit representation as a function over the set of possible deals and that for any deal its utility value can be calculated easily. We argue however, that a more realistic model of negotiations would be one in which the negotiator has certain knowledge about the domain and must reason with this knowledge in order to determine the value of a deal, which is time-consuming. We propose to use Game Description Language to model such negotiation scenarios, because this may enable us to apply existing techniques from General Game Playing to implement domain-independent, reasoning, negotiation algorithms.

Keywords

Automated Negotiation; General Game Playing; Game Description Language

1. INTRODUCTION

Most work on Automated Negotiations focuses purely on the strategy to determine which deals to propose *given* the utility values of the possible deals. Little attention has been given to negotiation settings in which determining the utility value of a deal is itself a hard problem that takes a substantial amount of time. One often assumes the utility value of any deal is known instantaneously, or can be determined by solving a simple linear equation [1]. In such studies the process of evaluating the proposal is almost completely abstracted away and one either assumes that the negotiation algorithms do not require any domain knowledge or reasoning at all, or that all such knowledge is hardcoded in the algorithm. The preferences of the agent's opponents on the other hand, are often assumed to be completely unknown.

In this paper however, we argue that in real negotiations it is very important to have domain knowledge, and a good negotiator must be able to reason about this knowledge. One cannot, for example, expect to make profitable deals in the antique business if one does not have extensive knowledge of antique, no matter how good one is at bargaining. Moreover, a good negotiator should also be able to reason about the desires of its opponents. A good car salesman for example would try to find out what type of car would best suit his client's needs, in order to increase the chances of coming to a successful deal.

We therefore propose a new kind of negotiation setting in which the agents do not have an explicit representation of their utility functions but instead are presented with domain knowledge in the form of a logic program. Agents will need to apply logical reasoning in order to determine the value of any proposal.

Another point that is rarely taken into account, is that an agent's utility may not always solely depend on the agreements it makes, but may also depend on decisions taken outside the negotiation thread. For example, suppose that you negotiate with a car salesman to buy a car. If you are single and you live in the city then it may be a very good deal to buy a small car which is easy to park and uses little fuel. However, if one year later you get married and decide to start a family, that deal suddenly is not so good anymore because you now require a larger family car. Interestingly, we see that although the deal itself has not changed at all, its utility value certainly has changed as a consequence of some decision taken long after the negotiations had finished.

Moreover, an agent's utility may not only depend on its own actions, but also on actions of other agents, as is typical for business deals. Imagine for example renting a property to open a restaurant in a street with no other restaurants. This might be a good deal until suddenly five other restaurants also open in that same street, giving you so much competition that you can no longer afford the rent.

We note that these properties we are addressing here– applying logical reasoning about the domain, and choosing a proper strategy with respect to your opponents' strategies– are also the main issues in the field of *General Game Playing* (GGP). General Game Playing deals with the implementation of agents that can play *any* kind of game. In contrast to specialized Chess- or Go- computers, which can only play one specific game and are largely based on knowledge provided by human experts, a GGP program cannot apply any game-specific heuristics because it only knows the rules of the games it is playing at run-time.

Therefore, in this paper we propose to use Game Description Language (GDL), which is commonly regarded as the standard language for GGP research [18], to define negotiation domains and we propose to use common techniques from GGP to implement negotiating agents. We investigate

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

to what extent GDL is applicable to the field of automated negotiations and compare its advantages and disadvantages. We conclude that describing negotiation domains in GDL is indeed possible, but that some small adaptations may need to be made to GDL to make it more suitable for negotiations.

Another advantage of using GDL for negotiations, is that it allows us to write protocol-independent agents. Currently, negotiating agents are often implemented for only one specific protocol. By applying GGP techniques we could be able to implement an agent that "understands" any kind of protocol as long as it is specified in GDL. So far we have indeed managed to specify the Alternating Offers protocol in GDL.

The rest of this paper is organized as follows. In Section 2 we give an overview of existing work in Automated Negotiations and General Game Playing. In Section 3 we explain how we can model a negotiation scenario as a game. In Section 4 we give a short description of GDL and in Section 5 we explain that for negotiation games specified in GDL we can distinguish between three types of agents. Then, in Section 6 we give a short description of a recently introduced language for strategic reasoning that can be used by negotiating agents to make proposals to each other. Next, in Section 7 we discuss some of the issues we encountered when applying GDL to negotiations. In Section 8 we present some preliminary results that we have so far obtained, and finally, in Section 9, we summarize our conclusions.

2. RELATED WORK

The earliest work on automated negotiations was mainly focused on highly idealized scenarios in which it is possible to formally prove certain theoretical properties, such as the existence of equilibrium strategies. A seminal paper in this area is the paper by Nash [21] in which he shows that under certain axioms the outcome of a bilateral negotiation is the solution that maximizes the product of the players' utilities. Many papers have been written afterwards that generalize or adapt some of his assumptions. A non-linear generalization has been made for example in [9]. Such studies give hard guarantees about the success of their approach, but the downside is that it is difficult to apply those results in real-world settings, since many of the assumptions made do not hold in the real world. A general overview of such game theoretical studies is made in [25].

In later work focus has shifted more towards heuristic approaches. Such work focuses on the implementation of negotiation algorithms for domains where one cannot expect to find any formal equilibrium results, or where such equilibria cannot be determined in a reasonable amount of time. It is usually not possible to give hard guarantees about the success of such algorithms, but they are more suitable to real-world negotiation scenarios. Important examples in this area are [7], and [8]. They propose a strategy that amounts to determining for each time t which utility value should be demanded from the opponent (the *aspiration level*). However, they do not take into account that one first needs to find a deal that indeed yields that aspired utility level. They simply assume that such a deal always exists, and that the negotiator can find it without any effort.

In general, these heuristic approaches still often make many simplifying assumptions. They may for example assume there is only a small set of possible agreements, or that the utility functions are linear additive functions which are explicitly given or which can be calculated without much computational cost. All these assumptions were made for example in the first four editions of the annual Automated Negotiating Agent Competition (ANAC 2010-2013) [1].

Recently, more attention has been given to more realistic negotiation settings in which the number of possible deals is very large so that one needs to apply search algorithms to find good deals to propose, and where utility functions are non-linear, for example in [19, 14, 20]. Although their utility functions are indeed non-linear over the vector space that represents the space of possible deals, the value of any given deal can still be calculated quickly by solving a linear equation. Even though in theory any non-linear function can indeed be *modeled* in such a way, in real-world settings utility functions are not always *given* in this way (e.g. there is no known closed-form expression for the utility function over the set of all possible configurations of a Chess game). In order to apply their method one would first need to transform the given expression of the utility function into the expression required by their model, which may easily turn out to be an unfeasible task.

Therefore, the idea of complex utility functions was taken a step further in [4], where the utility functions were not only non-linear, but determining the value of any deal was actually an NP-hard problem. Another important example of negotiations where determining utility values involves a hard combinatorial problem is the game of Diplomacy. In this game negotiations are even more complex because the utility values of the players are not directly defined in terms of the agreements they make, but more indirectly through the moves they make in the game. The players negotiate with one another about which moves each will make, which in turn influences the outcome of the game in a non-trivial manner. Determining the effect of an agreement on the player's final utility is a very hard problem that involves Game Theory and Constraint Satisfaction. Pioneering work on negotiations in Diplomacy was presented in [23, 17]. New interest in Diplomacy as a test-bed for negotiations has arisen with the development of the DipGame platform [6], which makes the implementation of Diplomacy agents easier for scientific research. Several negotiating agents have been developed on this platform [10, 5, 3].

General Game Playing is a relatively new topic. Although earlier work has been done, it really started to draw widespread attention in the AI community after the introduction of GDL [18] and the organization of the annual AAAI GGP competition since 2005 [12].

Common techniques applied by GGP players are minimax [27], alpha-beta pruning [15] and Monte Carlo Tree Search (MCTS) [16]. All these techniques generate a search tree in which each node ν represents a certain state w_ν of the game and a certain player α_ν. The root node represents the initial state, and for each node the edges to its child nodes represent the actions that are legal in the state w_ν, for player α_ν. Each time a new node ν is added to the tree, the algorithm parses the game rules, which are written in GDL, to determine which actions are legal for player α_ν in the state w_ν and, if w_ν is a terminal state, which utility value each player receives.

FluxPlayer [24], the winner of the 2006 AAAI GGP competition applies an iterated deepening depth-first search method with alpha-beta pruning, and uses Fuzzy logic to determine

how close a given state is to the goal state. Cadia Player [11], the winner in 2007, 2008, and 2012, is based on MCTS extended with several heuristics to guide the playouts so that they are better informed and hence give more realistic results. Furthermore, also the winner of 2014, Sancho[1], as well as the winner of 2015, Galvanise[2] apply variants of MCTS.

3. NEGOTIATION GAMES

Since GDL is a language to describe games, in this section we explain how a negotiation scenario can be described as a game.

Game theory can be related to negotiations in two possible ways. Firstly, the negotiation protocol can be modeled as a game. This approach is for example taken in Nash' famous paper [21] and is the common approach taken when one intends to formally prove properties of a negotiation scenario. In this case the moves made by the players consist of making proposals and accepting proposals, and the utility functions are directly given as a function over the space of possible outcomes of the negotiation protocol.

However, in this paper we follow a new approach in which not only the protocol, but also the utility functions are defined by means of a Game Theoretical model. That is: players receive utility by making certain moves in some game G, and on top of that they are allowed to negotiate about the moves they will make in that game according to some negotiation protocol N. A typical example of such a negotiation scenario is the game of Diplomacy. In this case there are two types of moves: negotiation-moves (i.e. making proposals, accepting proposals or rejecting proposals) and game-moves (the moves defined in the game G). The proposals that players make or accept are proposals about which game-moves they will make. The utilities of the players are only determined by the game-moves. However, since the agreements they make during the negotiations will partially restrict their possible game-moves, the utility values obtained by the players indirectly do depend on the negotiated agreements.

We will first define the concept of a *protocol*, and then define the concepts of a *negotiation protocol* and of a *game*, which are two different extensions of the concept of a protocol. Next, we will define a *negotiation game*, which is a combination of a negotiation protocol and a game.

DEFINITION 1. *A **protocol** P is a tuple $\langle Ag, \mathcal{A}, W, w_1, T, L, u\rangle$, where:*

- *Ag is the set of **agents** (or **players**): $Ag = \{\alpha_1, \alpha_2, \dots \alpha_n\}$*
- *$\mathcal{A}$ is a tuple $(\mathcal{A}_1, \mathcal{A}_2, \dots \mathcal{A}_n)$ where each $\mathcal{A}_i$ is the set of **actions** (or **moves**) of agent α_i.*
- *W is a non-empty set of **states**.*
- *$w_1 \in W$ is the **initial** state.*
- *$T \subset W$ is the set of **terminal** states.*
- *L is a tuple $(L_1, L_2, \dots L_n)$, where each L_i is the **legality** function for α_i, which assigns to each non-terminal state a nonempty set of actions for α_i. $L_i : W \setminus T \to 2^{\mathcal{A}_i}$.*
- *$u : W \times \mathcal{A}_1 \times \mathcal{A}_2 \cdots \times \mathcal{A}_n \to W$ is the update function that maps each state and action profile to a new state.*

We say an action a is a **legal action** for player α_i in state w iff $a \in L_i(w)$. A tuple $\vec{a} = (a_1, a_2, \dots a_n)$ consisting of one action $a_i \in \mathcal{A}_i$ for each player is called an **action profile**. An action profile is called a **legal action profile** in state w iff all actions a_i of that action-profile are legal in w. Given a state w and a legal action profile $\vec{a}$ the update function u defines the **next state** w' as: $w' = u(w, \vec{a})$. A **legal history** is a finite sequence of states, $(w_1, w_2, \dots w_m)$, starting with the initial state w_1 such that for each integer j with $1 \leq j < m$ there exists a legal action-profile $\vec{a}$ such that $u(w_j, \vec{a}) = w_{j+1}$.

Note that in this model it is assumed that the agents always take their actions simultaneously. This is not really a restriction because any turn-taking protocol can be modeled as a special case of a simultaneous-move protocol, by adding a special dummy-move, often called 'noop', to the model which has no effect on the next state. Then, one can define the legality functions such that in every state all players except one have only one legal move, which is the 'noop' move. The one player that does have more than one legal move is then called the **active player** of that state.

DEFINITION 2. *A **negotiation protocol** N is a tuple $\langle P, Agr, C, \eta\rangle$, where:*

- *P is a protocol.*
- *Agr is a nonempty set of possible agreements, known as the **agreement space**.*
- *$C : T \to Agr$ is the **commitment function** that maps every terminal state of the protocol to an agreement.*
- *$\eta \in Agr$ is the 'conflict deal'.*

The set Agr can be any set that represents the possible deals the agents can make with each another. The interpretation of C is that if the negotiation protocol ends in a terminal state w then $C(w)$ is the agreement that the agents have agreed upon.[3] The set Agr contains one element η that represents the 'conflict deal' i.e. an element to represent that no deal has been made. So if the agents do not come to any agreement, than the protocol ends in a final state w for which $C(w) = \eta$.

As an example, let us define the alternating offers protocol [22] using this model.

EXAMPLE 1. *Suppose we have two agents negotiating how to split a pie according to an alternating offers protocol over m rounds. The agents are denoted $Ag = \{\alpha_1, \alpha_2\}$. The possible agreements are the real values between 0 and 1, representing the fraction of the pie assigned to player α_1, so: $Agr = [0, 1] \cup \{\eta\}$. The actions of the players are either to propose a division of the pie, or to accept the previous proposal, or to do nothing:*

$$\mathcal{A}_1 = \mathcal{A}_2 = \{propose(x) \mid x \in [0, 1]\} \cup \{accept, noop\}.$$

A state is given as a triple: (r, x, b) where r is the round of the protocol, x is the last proposal made, and b is either 'true' ($\top$) or 'false' ($\bot$) indicating whether x has been accepted or not.

$$W = \{(r, x, b) \mid 0 \leq r \leq m, \quad x \in Agr, \quad b \in \{\top, \bot\}\}$$

[1] http://sanchoggp.blogspot.co.uk/2014/05/what-is-sancho.html

[2] https://bitbucket.org/rxe/galvanise_v2

[3] We could generalize this and allow protocols in which more than one deal can be made. However, we will not do so here for simplicity.

The initial state is: $w_1 = (0, \eta, \bot)$. *Terminal states are those states in which either the last round has passed or any of the agents has accepted a proposal:*

$$T = \{(r, x, b) \in W \mid r = m \quad \vee \quad b = \top\}$$

In the even rounds player α_1 *is the active player and in the odd rounds* α_2 *is active. In every state all actions except 'noop' are legal for the active player, except that in the initial state it is also not allowed to play 'accept' (because no proposal has yet been made that could be accepted).*
If $r = 0$*:*

$$L_1(r, x, b) = \mathcal{A}_1 \setminus \{accept, noop\} \quad L_2(r, x, b) = \{noop\}$$

If $r > 0$*:*

$$L_i(r, x, b) = \mathcal{A}_i \setminus \{noop\} \qquad L_j(r, x, b) = \{noop\}$$

with $i = r \pmod 2 + 1$ *and* $j \neq i$. *The update function is defined as follows:*

$$\begin{aligned} u((r, x, \bot), propose(y), noop) &= (r + 1, y, \bot) \\ u((r, x, \bot), noop, propose(y)) &= (r + 1, y, \bot) \\ u((r, x, \bot), accept, noop) &= (r + 1, x, \top) \\ u((r, x, \bot), noop, accept) &= (r + 1, x, \top) \end{aligned}$$

And finally, the commitment function returns the proposal that was accepted or, if no proposal was accepted, returns the conflict deal:

$$C(r, x, \top) = x \quad C(m, x, \bot) = \eta$$

Note that this definition of the alternating offers protocol can be adapted easily to domains other that split-the-pie, simply by replacing Agr by some other agreement space. Everything else remains the same.

Definition 3. *A* ***game*** G *is a pair* $\langle P, U \rangle$ *where* P *is a protocol and* U *is a tuple* $U = (U_1, U_2, \ldots U_n)$ *where each* U_i *is the* ***utility function*** *of player* α_i*, which assigns a utility value to each terminal state of the protocol:* $U_i : T \to \mathbb{R}^+$.

The goal of each player α_i is to choose a strategy such that the game ends in a terminal state w_m that maximizes his or her utility $U_i(w_m)$. As an example, let us define the prisoner's dilemma using this model.

Example 2. *If* G *is the prisoner's dilemma then we have the following protocol (c stands for 'confess' and d stands for 'deny'):*

- $Ag = \{\alpha_1, \alpha_2\}$
- $\mathcal{A}_1 = \mathcal{A}_2 = \{c, d\}$
- $W = \{w_1, w_{cc}, w_{cd}, w_{dc}, w_{dd}\}$
- $T = \{w_{cc}, w_{cd}, w_{dc}, w_{dd}\}$
- $L_1(w_1) = L_2(w_1) = \{c, d\}$
- $u(w_1, c, c) = w_{cc}, \quad u(w_1, c, d) = w_{cd},$
  $u(w_1, d, c) = w_{dc}, \quad u(w_1, d, d) = w_{dd}$

Here, w_1 *is the initial state,* w_{cc} *is the terminal state that is reached when both players play c,* w_{cd} *is the terminal state that is reached when* α_1 *plays c and* α_2 *plays d, etcetera. The utility functions can, for example, be defined as:*

- $U_1(w_{cc}) = U_2(w_{cc}) = 2$
- $U_1(w_{cd}) = U_2(w_{dc}) = 10$
- $U_1(w_{dc}) = U_2(w_{cd}) = 0$
- $U_1(w_{dd}) = U_2(w_{dd}) = 8$

Definition 4. *Given a game* G*, a* ***strategy*** σ *for player* α_i *is a map that maps every non-terminal state of that game to a nonempty set of legal moves for that player. Thus,* σ *is a map:*

$$\sigma : W \setminus T \to 2^{\mathcal{A}_i}$$

such that for each $w \in W \setminus T$ *we have* $\sigma(w) \neq \emptyset$ *and* $\sigma(w) \subseteq L_i(w)$. *A* ***complete strategy*** *is a strategy such that* $|\sigma(w)| = 1$ *for all* $w \in W \setminus T$*, and a* ***partial strategy*** *is a strategy that is not complete. A tuple* $(\sigma_1, \sigma_2 \ldots \sigma_n)$ *consisting of one strategy for each player is called a* ***strategy profile****.*

In the following, we use a superscript G or N to indicate that something is a component of the game G or of the negotiation protocol N. For example P^G is the protocol of G, and W^N is the set of world states of N.

We will next define a negotiation game NG to be a combination of a negotiation protocol N and a game G. The interpretation of NG is that it is a game that consists of two stages: a negotiation stage followed by an action stage. In the action stage the players play the game G, while in the preceding negotiation stage the players negotiate about which strategies they will apply during the action stage. These agreements are considered binding, therefore, if the players come to an agreement they will have less legal moves during the action stage than they would have in the pure game G.

We say a negotiation protocol N is compatible with a game G if it has the same set of agents as G, and the set of agreements Agr consists purely of strategy profiles for the game G. This means that N is designed for the agents of G to negotiate the strategies they will play in the game G.

Definition 5. *A negotiation protocol* N *is* ***compatible*** *with a game* G*, if both of the following hold:*

- $Ag^N = Ag^G$.
- *If* $x \in Agr^N$ *then* x *is a strategy profile for* G.

The interpretation here, is that if the negotiators agree on some strategy profile $(\sigma_1, \ldots \sigma_n)$ then each player α_i has promised, for any state w of G, to only choose its action from $\sigma_i(w)$. Specifically, if σ_i is a *complete* strategy, then α_i has no more free choice in G, and must play in any state w the unique action in $\sigma_i(w)$. Furthermore, the conflict deal of Agr^N corresponds to the strategy profile in which each player still has its full set of legal actions to choose from: $\sigma_i(w) = L_i(w)$. Indeed, if no agreement is made this means that no agent is restricted by any commitments and may therefore choose any legal action in G.

Definition 6. *Given a game* G *and a negotiation protocol* N *compatible with* G *we define the* ***negotiation game*** NG *as a game, such that:*

- $Ag = Ag^N = Ag^G$
- *For each player* α_i*:* $\mathcal{A}_i = \mathcal{A}_i^N \cup \mathcal{A}_i^G$
- *The set of states* W *is a subset of* $W^N \times W^G$.
  More precisely: $W = W^{nego} \cup W^{act}$ *with:*

$$\begin{aligned} W^{nego} &= W^N \times \{w_1^G\} \\ W^{act} &= T^N \times W^G \end{aligned}$$

- *The initial state is defined as:* $w_1 = (w_1^N, w_1^G)$.

- *The terminal states are defined as:* $T = T^N \times T^G$.
- *The update function is defined as:*

$$\begin{aligned} u((v,z),\vec{a}) &= (u^N(v,\vec{a}),z) \quad \text{if } (v,z) \in W^{nego} \\ u((v,z),\vec{a}) &= (v,u^G(z,\vec{a})) \quad \text{if } (v,z) \in W^{act} \end{aligned}$$

- *The legality functions are defined as:*

$$\begin{aligned} L_i(v,z) &= L_i^N(v) \quad \text{if } (v,z) \in W^{nego} \\ L_i(v,z) &= \sigma_i(z) \quad \text{if } (v,z) \in W^{act} \end{aligned}$$

  where $(\sigma_1, \ldots \sigma_n) = C^N(v)$
- *The utility functions are defined as:*

$$U_i(v,z) = U_i^G(z)$$

Here, we have modeled states w of NG as pairs of states $w = (v,z)$ consisting of a protocol-state $v \in W^N$ and a game-state $z \in W^G$. The initial state w_1 of NG is simply the pair of initial states (w_1^N, w_1^G) of N and G.

We have divided the state space W into two subspaces: W^{nego} and W^{act}, which represent the negotiation stage and the action stage respectively. Note that the initial state is in W^{nego} and that the update and legality functions are defined such that any legal history starts with a sequence of states that are all in W^{nego}, followed by a sequence of states that are all in be in W^{act}. In other words: the game starts in the negotiation stage, until at some point it reaches a state in the action stage, after which the game remains in the action stage.

During the negotiation stage the update function u only acts on the protocol-state, and acts on it according to the update function u^N of N, while during the action stage u only acts on the game-state, according to u^G. In other words: during the negotiation stage the game follows the protocol P^N, while during the action stage the game follows the protocol P^G.

Similarly, during the negotiation stage the legality functions L_i are simply the legality functions L_i^N of the negotiation protocol. Therefore, during this stage the agents can make proposals and accept proposals. During the action stage the legality function of an agent α_i allows it only to take those actions it was committed to during the negotiation stage. Note that indeed, if $(v,z) \in W^{act}$ then v is a terminal state of N, and therefore $C^N(v)$ is some agreement from the agreement space Agr^N of N. Furthermore, since N is compatible with G, we know that $C^N(v)$ is a strategy profile of G. In other words: if during the negotiation stage the agents have agreed to on the strategy profile $C^N(v) = (\sigma_1, \ldots \sigma_n)$ then during the action stage each agent α_i is committed to play according to the strategy σ_i.

EXAMPLE 3. *If G is the Prisoner's Dilemma, and N is the alternating offers protocol compatible with G, then the* ***Negotiating Prisoner's Dilemma*** *NG begins with a negotiation stage in which the prisoners may negotiate which strategies they will play. The prisoners may propose any strategy-pair (σ_1, σ_2). Since there is only one non-terminal state in the Prisoner's Dilemma, a strategy is defined by its value $\sigma_i(w_1^G)$ on that non-terminal state w_1^G. That is, σ_i can be either $\sigma_i(w_1^G) = \{c\}$ or $\sigma_i(w_1^G) = \{d\}$ or $\sigma_i(w_1^G) = \{c,d\}$. If the prisoners are rational, then one of them will propose the strategy profile $(\{d\},\{d\})$ and the other will accept that proposal. In the action stage they are then both committed to play the action d.*

Note that this example, in which the players agree to play $(\{d\},\{d\})$, is in fact a subgame perfect equilibrium of the Negotiating Prisoner's Dilemma.[4] This is interesting, because the outcome strictly dominates the outcome (c,c) which is the Nash-equilibrium of the pure Prisoner's Dilemma. Therefore, in a sense, we can say that we have 'solved' the Prisoner's Dilemma by giving the players the opportunity to negotiate and make binding agreements about their actions.

Finally, we would like to remark that in the definition of a Negotiation Game as presented in this section the players only have one opportunity to negotiate, before they play the game G. However, we could also consider more general models in which, for example, the players have a new opportunity to negotiate before each new turn of the game G. We will not do this however, to keep the discussion simple.

4. GAME DESCRIPTION LANGUAGE

In this section we will give a short introduction to GDL. For more details we refer to [13].

GDL is logical language that was designed to describe games. In principle, it can describe any game G defined according to Definitions 1 and 3. GDL is similar to Datalog [2], but it defines the following relation symbols:[5] *init*, *true*, *next*, *legal*, *goal*, *terminal*, *does*, which have a special meaning related to games.

In GDL a state w of a game is represented as a set of atomic formulas, which we will here denote as $V(w)$. These atoms are all of the form $true(p)$, where p can be any ground term. For example, in Tic-Tac-Toe the state in which the the center cell contains the marker X and the left upper cell contains the marker O could be represented as:

$$V(w) = \{\ true(cell(2,2,X))\ \ ,\ \ true(cell(1,1,O))\ \}$$

A GDL **rule** is an expression of the following form:

$$s_1 \wedge s_2 \wedge \ldots s_n \rightarrow h$$

where each s_i is a positive or negative literal, and h is a positive literal. The atom h is called the *head* of the rule and the s_i's are called the *subgoals* of the rule. The conjunction of subgoals is called the *body* of the rule. The body of a rule may be the empty conjunction, in which case the rule is also referred to as a *fact*, which we denote as $\rightarrow h$.

A game description is then nothing more then a set of GDL rules. For example, if the game description contains the following rule:

$$true(p) \wedge does(\alpha_i, a) \rightarrow next(q)$$

it means that if the game is in a state w for which $true(p) \in V(w)$ and player α_i plays action a then in the next round the game will be in a state w' for which $true(q) \in V(w')$ holds. Similarly:

$$true(p) \rightarrow terminal$$

means that any state w for which $true(p) \in V(w)$ holds is a terminal state. The fact

$$\rightarrow init(p)$$

[4]We should stress here that we have assumed agreements are *binding*. Without this assumption this statement would not be true.

[5]GDL defines more relations, but these are not relevant for this paper.

means that for the initial state w_1 we have $true(p) \in V(w_1)$. The rule

$$true(p) \rightarrow legal(\alpha_i, a)$$

means that for any state in which $true(p) \in V(w)$ holds it is legal for player α_i to play the move a.

$$true(p) \rightarrow goal(\alpha_i, 100)$$

means that in any state w for which $true(p) \in V(w)$ holds α_i receives a utility value of 100.

GDL uses what is known as negation-by-failure. This means that a negative literal $\neg p$ is considered true if and only if there is no rule from which one can derive the truth of p. GDL can only describe games of full information without randomness. However, an extension to GDL exists, called GDL-II [26] which does allow for randomness and private information.

The Game Manager is a server application specially designed for General Game Playing. It allows you to start a game with a game description written in GDL, and allows game playing agents to connect and start playing. Once connected, the server sends the game description to the players. The players then need to parse the description and determine for themselves which moves they can make, and what the consequences are. Every round, each player is supposed to send a message back with the move it desires to play. If a player fails to send this message, or if it chooses an action that is illegal in the current state, the server will instead pick a random move for that player. Next, the server sends a message back to all players, informing each player which moves have been made by the other players. From this information the players can then compute the next state, and determine which moves they can make in that new state. This continues until a terminal state is reached.

5. GENERAL NEGOTIATION GAMES

Not only does GDL allow us to define complex negotiation games, it also allows us to write domain-independent negotiation algorithms. With this we mean that the agents do reason about the domain, but they only receive information about the domain at run-time, just like in GGP.

We can now distinguish between three types of agents:

- **Completely generic agents**: agents that are able to interpret any negotiation protocol N and any game G provided at run-time, as long as they are specified in GDL.
- **Protocol-specific agents**: agents that are designed only for one specific negotiation protocol N, but that can interpret any game G provided at run-time, as long as it is specified in GDL.
- **Game-specific agents**: agents that are able to interpret any negotiation protocol N provided at run-time, as long as it is specified in GDL, but that are designed only for one specific game G.

Agents that are completely generic are essentially just GGP agents. Since they can handle any negotiation protocol N and any game G they can handle in principle anything specified in GDL, including games that have nothing to do with negotiations. This, however, also means that when such an agent is playing a negotiation game, it is very hard (if not impossible) for this agent to exploit that fact, and be better at negotiating than any standard GGP algorithm. Therefore, we think that this kind of agent is less interesting for Automated Negotiations research.

We think that protocol-specific agents are more interesting. Such an agent is less generic than a GGP player, but it has the advantage that when implementing it we may incorporate algorithms specific for negotiations. In order to assess the values of the proposals it negotiates it would need to apply advanced reasoning algorithms for GGP, which makes it more interesting than the agents developed for classical negotiation domains.

Another interesting option is to implement game-specific agents. This would allow us to do research on negotiation algorithms that are independent of the negotiation protocol.

6. A LANGUAGE TO DEFINE STRATEGIES

As explained, the proposals made during the negotiation stage of a Negotiation Game are in fact strategy profiles. For example, in the Negotiating Prisoner's Dilemma, player α_1 may propose:

$$(\sigma_1, \sigma_2) = (\{c, d\}, \{d\})$$

Here, σ_1 is the partial strategy in which player α_1 has the choice to play either c or d, and σ_2 is the strategy in which α_2 plays d (of course this is a highly unprofitable deal for α_2 so if α_2 is rational he or she will not accept it).

In the case of the Prisoner's Dilemma a strategy can be represented simply as the set of possible actions in the only non-terminal state. However, in other games, such as Diplomacy, the number of possible actions and states can be extremely large. Therefore, expressing a strategy explicitly as a set of actions for every non-terminal state in such games is infeasible.

Instead, we propose to use Strategic Game Logic: a recently introduced logical language specifically defined to describe game-strategies. SGL is in fact an extension of GDL. While GDL is used to describe the *rules* of a game, SGL can be used to describe *strategies* of a game. We will here only briefly discuss the basic ideas. For a detailed description we refer to [28].

A logical formula ϕ in SGL may represent a set of states, or a set of state-action pairs. For example: $\phi = true(p) \wedge true(q)$ would represent the set of states:

$$\{w \in W \mid true(p) \in V(w) \text{ and } true(q) \in W\}$$

while $\phi = true(p) \wedge does(a)$ represents the set of state-action pairs:

$$\{(w, a) \in W \times \{a\} \mid true(p) \in V(w)\}$$

We say that w satisfies ϕ if w is in the set of states represented by ϕ, and we say that (w, a) satisfies ϕ if (w, a) is in the set of state-action pairs represented by ϕ.

Note that a set S of state-action pairs can be seen as a strategy, defined by $a \in \sigma_i(w)$ iff $(w, a) \in S$. Therefore, negotiators may make proposals of the form $propose(\phi_1, \phi_2)$ where ϕ_1 is an SGL formula that represents a strategy for player α_1 and ϕ_2 is an SGL formula that represents a strategy for player α_2.

SGL defines a number of new operators on top of GDL. That is, if a is an action and ϕ and φ are formulas, then SGL defines the following expressions: $[a]\phi$, $\lfloor a \rfloor \phi$, and $\wr \varphi \wr \phi$.

Let ϕ be any formula representing a set of states, and a be any action. If w satisfies ϕ, then by definition $u(a, w)$

satisfies $[a]\phi$. That is: $[a]\phi$ represents the set of states that result from the action a being played in a state that is represented by ϕ. Furthermore, w satisfies ϕ if and only if (w, a) satisfies $\lfloor a \rfloor \phi$.

For the last operator we have that $\wr \varphi \wr \phi$ is satisfied only if ϕ is satisfied under a new protocol where the legality relation of the original protocol is replaced by the strategy φ. For example, suppose that we have a state w in which it is legal for player α_i to play either action a or action b. Then the formulas $\phi_1 = legal(\alpha_i, a)$ and $\phi_2 = legal(\alpha_i, b)$ are both satisfied by w. Furthermore, let φ be a strategy in which α_i plays action b in state w. Then we have that w satisfies $\wr \varphi \wr \phi_2$, but not $\wr \varphi \wr \phi_1$.

Using these new operators SGL defines two more operators that allow to combine any two strategies into a new strategy. Firstly, SGL defines *prioritized disjunction*:

$$\phi \oslash \varphi = \phi \vee (\varphi \wedge \bigwedge_{c \in \mathcal{A}_i} \lfloor c \rfloor \neg \phi)$$

which has the interpretation of *"play strategy ϕ if applicable, otherwise play strategy φ"*. Secondly, SGL defines *prioritized conjunction*:

$$\phi \otimes \varphi = \phi \wedge ((\bigvee_{c \in \mathcal{A}_i} \lfloor c \rfloor (\phi \wedge \varphi)) \rightarrow \varphi)$$

Which is a strategy with the interpretation: *"apply both ϕ and φ if both are applicable; otherwise, apply ϕ only"*.

7. APPLYING GDL TO NEGOTIATIONS

In this section we will discuss some technical issues that we encountered when using GDL to describe negotiation games.

7.1 Enforcement of Commitments

One problem we need to take care of is the question how to enforce that agents obey their agreements. Although in some domains (such as Diplomacy) there simply is no mechanism at all to force agents to obey agreements, in many existing domains one does require agreements to be enforced. We suggest two possible solutions.

The first option would be to write a game description that includes all rules for the negotiation protocol N, the game G and all rules necessary to guarantee agreement obedience. That is, they would contain rules of the form "If players α_1 and α_2 make the agreement that α_1 will play action a_1 and α_2 will play action a_2, then those actions will be the only legal actions". For example, if the original game G contains the rule

$$true(p) \rightarrow legal(\alpha_i, a)$$

then the negotiation game NG would instead contain the rules:

$$committed(\alpha_i, a) \wedge a \neq b \rightarrow excluded(\alpha_i, b)$$
$$true(p) \wedge \neg excluded(\alpha_i, a) \rightarrow legal(\alpha_i, a)$$

where the first rule specifies that whenever a player gets committed to an action a, then all other actions are excluded. The second rule is an adaptation of the original rule, with the extra premise added that an action a can only be legal if it has not been excluded by the commitments.

The second option would be to write a new game server that forbids the players to make moves that are incompatible with their agreements. In that case we see the negotiation protocol N and the game G as two separate games with each its own game description. If a player gets committed to action a but tries to play the action b the server will not allow it, even though b is legal according to the rules of the pure game G.

The advantage of the first option is that it is completely compatible with existing GGP standards. A negotiation game NG is just another game that can be described in GDL. Any existing GGP player should therefore be able to participate in such a negotiation game. However, the problem is that one would need to write rules that take all possible commitments into account. After all, a commitment may not simply be a single action, as in this example, but could be a disjunction of actions, or it could be conditional (e.g. if you play a I will play b in the next round, or if you play b then I will play a). It would be very complicated to write rules that are generic enough to cover all possible commitments, especially if we allow the full SGL language to specify agreements. Furthermore, it seems rather redundant to explicitly write rules to enforce commitments, if it is obvious that agreements must be obeyed.

We therefore think it might be more practical to choose the second option and implement a special General Negotiation server that handles rule enforcement. Moreover, it also has the advantage that it would allow us to re-use any existing game specification and freely combine it with any negotiation protocol specified in GDL. There is no need to adapt the rules of the game.

7.2 Very Large Spaces

Currently, many GGP algorithms are not able to handle domains where the number of possible actions is very large. The reason is that they apply *grounding*: they try to generate a list that explicitly contains all possible actions. Of course, if there are millions of possible proposals such as in the domain of the ANAC 2014 competition, this approach will not work.

However, we have managed to implement a GDL specification of a domain like in ANAC 2014, where we avoid this problem with a little trick. In this domain the negotiators propose contracts that consist of 10 properties, and each property can have 10 different values, so that there are 10^{10} possible contracts.

Instead of mapping each possible proposal to an action, we have written the description such that making a proposal requires making several actions. More precisely: apart from the three standard types of action: 'propose', 'accept' and 'noop' from Example 1, we have added a fourth action called 'setValue'. The setValue action takes two parameters: a property-index and a value. By playing a number of setValue actions the player creates a contract, in which the indicated properties have the indicated values. For example, if an agent plays the following three actions:

$$setValue(1, 5), \quad setValue(2, 9), \quad setValue(4, 2)$$

it creates a contract in which the first property has value 5, the second property has value 9 and the fourth property has value 2. All other properties will by default have value 0. Then, after generating the contract the negotiator can propose it by playing the action 'propose'. In this way there are only 103 possible actions, instead of 10^{10}.

7.3 Continuous Time

In GDL it is assumed that games take place over discrete rounds. The duration of each round can be specified in the server. However, in negotiations it is not uncommon to assume that negotiations take place in continuous time. In the ANAC 2014 domain for example, although the agents did take turns, each agent could take as much time as it wanted to make a proposal. In principle, this is not a problem, because we can simple allow agents to make a 'noop' move representing the 'action' of not making an action, and make sure that the other agent only gets the turn after the first agent makes a proposal.

There is however a small technical problem with this, namely that each agent must take care that it indeed submits the 'noop' action before the deadline of the round passes. If it does not manage to do this in time (for example because it is doing heavy calculations that take up all its resources) then the server will automatically pick a random action for that agent. Of course, this is undesirable because the server may choose a highly unprofitable proposal which may then be accepted by the opponent.

We think that, in the context of negotiations in continuous time, it would be better to have a server that by default picks the 'noop' move if you fail to play any action within the time limits.

7.4 Hidden Utilities

In GDL it is not possible to specify games with hidden information. This means that you cannot only determine your own utility values, but also your opponents' utility values. This is fine for most games, but in the field of Automated Negotiations it is often assumed that utility values are hidden.

Again, in order to solve this problem we could write an alternative game server that does not send information to the players about their opponents' utility functions, or alternatively we might use GDL-II to keep information about utility functions hidden.

Another option, is to simply re-interpret the semantics of GDL. That is, we could interpret the goal-values specified in the game descriptions not as utility values, but rather as values that only indicate a preference *order*. For example, suppose we have the following two rules:

$$\begin{aligned} true(p) &\rightarrow goal(\alpha_1, 100) \\ true(q) &\rightarrow goal(\alpha_1, 50) \end{aligned}$$

The classical interpretation of this is the following: "if p is true, then α_1 receives a utility of 100, and if q is true then α_1 receives a utility of 50." However, we could re-interpret this as only meaning the following: "α_1 prefers states in which p is true over states in which q is true". In this second interpretation the values 50 and 100 do not really have any meaning any more. They only serve to establish an ordering between terminal states, while the true utility values remain private information.

8. RESULTS

We have managed to specify the Negotiating Prisoner's Dilemma of Example 3 in GDL, in which the negotiation stage was modeled as a turn-taking protocol with three rounds in which prisoner 1 is first allowed to make a proposal, next prisoner 2 is allowed to either accept that proposal or make a counter proposal, and finally prisoner 1 may accept the last proposal made by prisoner 2.

We have implemented a straightforward minimax algorithm for GGP, and when we let two instances of this algorithm play the Negotiating Prisoner's Dilemma they indeed successfully negotiate and agree to both play 'deny'. This is very interesting, because this algorithm is *not* a negotiation algorithm. It is simply a general game-playing algorithm that may just as well play Tic-Tac-Toe or any other simple game. The reason that it is able to negotiate successfully is that the negotiation scenario was described in GDL.

Moreover, we have implemented a domain similar to ANAC 2014 in which the agents negotiate according to the alternating offers protocol over a space with 10^{10} possible contracts. Since this is a very large domain over many rounds a naive minimax does not work. To be able to handle such domains we need to apply more state-of-the-art GGP techniques. We leave this as future work. Also it would be interesting to see whether any of the top existing GGP players is able to handle this domain.

9. CONCLUSIONS

We conclude that GDL is in essence a good option for the description of general negotiation scenarios because it allows us to write complex negotiation domains that require reasoning and logic, and for which assessing the value of a proposal requires thinking ahead about your future actions, as well as the opponents' future actions. Moreover, GDL allows us to write domain-independent agents, in the sense that they only receive domain-knowledge at run-time.

However, there are a number of aspects specific to automated negotiations that are not handled well by GDL and the existing GGP server. Therefore, we think that it is necessary to write a new server application, specific for negotiations. This server will handle rule enforcement and should be able to verify whether any action is compatible with any strategy defined as a formula in SGL.

We have shown that a simple Negotiating Prisoner's Dilemma can be described correctly in GDL, as well as the more complex domain of ANAC 2014. Furthermore, we have shown that it is indeed possible for a GGP algorithm to successfully negotiate in the Negotiating Prisoner's Dilemma even though it is not designed for negotiations. For the larger ANAC 2014 domain we still need to find out whether existing GGP techniques are able to handle it.

10. ACKNOWLEDGMENTS

This work was sponsored by an Endeavour Research Fellowship awarded by the Australian Government, Department of Education.

11. REFERENCES

[1] T. Baarslag, K. Hindriks, C. M. Jonker, S. Kraus, and R. Lin. The first automated negotiating agents competition (ANAC 2010). In T. Ito, M. Zhang, V. Robu, S. Fatima, and T. Matsuo, editors, *New Trends in Agent-based Complex Automated Negotiations, Series of Studies in Computational Intelligence*. Springer-Verlag, 2010.

[2] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to

ask). *IEEE Transactions on Knowledge and Data Engineering*, 1(1):146–166, 1989.

[3] D. de Jonge. *Negotiations over Large Agreement Spaces*. PhD thesis, Universitat Autònoma de Barcelona, 2015.

[4] D. de Jonge and C. Sierra. NB3: a multilateral negotiation algorithm for large, non-linear agreement spaces with limited time. *Autonomous Agents and Multi-Agent Systems*, 29(5):896–942, 2015.

[5] A. Fabregues. *Facing the Challenge of Automated Negotiations with Humans*. PhD thesis, Universitat Autònoma de Barcelona, 2012.

[6] A. Fabregues and C. Sierra. Dipgame: a challenging negotiation testbed. *Engineering Applications of Artificial Intelligence*, 2011.

[7] P. Faratin, C. Sierra, and N. R. Jennings. Negotiation decision functions for autonomous agents. *Robotics and Autonomous Systems*, 24(3-4):159 – 182, 1998. Multi-Agent Rationality.

[8] P. Faratin, C. Sierra, and N. R. Jennings. Using similarity criteria to make negotiation trade-offs. In *International Conference on Multi-Agent Systems, ICMAS'00*, pages 119–126, 2000.

[9] S. Fatima, M. Wooldridge, and N. R. Jennings. An analysis of feasible solutions for multi-issue negotiation involving nonlinear utility functions. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, pages 1041–1048, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[10] A. Ferreira, H. Lopes Cardoso, and L. Paulo Reis. Dipblue: A diplomacy agent with strategic and trust reasoning. In *7th International Conference on Agents and Artificial Intelligence (ICAART 2015)*, pages 398–405, 2015.

[11] H. Finnsson. *Simulation-Based General Game Playing*. PhD thesis, School of Computer Science, Reykjavik University, 2012.

[12] M. Genesereth, N. Love, and B. Pell. General game playing: Overview of the aaai competition. *AI Magazine*, 26(2):62–72, 2005.

[13] M. R. Genesereth and M. Thielscher. *General Game Playing*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2014.

[14] T. Ito, M. Klein, and H. Hattori. A multi-issue negotiation protocol among agents with nonlinear utility functions. *Multiagent Grid Syst.*, 4:67–83, January 2008.

[15] D. E. Knuth and R. W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293 – 326, 1975.

[16] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *Proceedings of the 17th European Conference on Machine Learning*, ECML'06, pages 282–293, Berlin, Heidelberg, 2006. Springer-Verlag.

[17] S. Kraus. Designing and building a negotiating automated agent. *Computational Intelligence*, 11:132–171, 1995.

[18] N. Love, M. Genesereth, and T. Hinrichs. General game playing: Game description language specification. Technical Report LG-2006-01, Stanford University, Stanford, CA, 2006. http://logic.stanford.edu/reports/LG-2006-01.pdf.

[19] I. Marsa-Maestre, M. A. Lopez-Carmona, J. R. Velasco, and E. de la Hoz. Effective bidding and deal identification for negotiations in highly nonlinear scenarios. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, pages 1057–1064, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[20] I. Marsa-Maestre, M. A. Lopez-Carmona, J. R. Velasco, T. Ito, M. Klein, and K. Fujita. Balancing utility and deal probability for auction-based negotiations in highly nonlinear utility spaces. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, pages 214–219, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[21] J. Nash. The bargaining problem. *"Econometrica"*, "18":155–162, 1950.

[22] J. S. Rosenschein and G. Zlotkin. *Rules of Encounter*. The MIT Press, Cambridge, USA, 1994.

[23] E. E. S. Kraus, D. Lehman. An automated diplomacy player. In D. Levy and D. Beal, editors, *Heuristic Programming in Artificial Intelligence: The 1st Computer Olympia*, pages 134–153. Ellis Horwood Limited, 1989.

[24] S. Schiffel and M. Thielscher. M.: Fluxplayer: A successful general game player. In *In: Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 1191–1196. AAAI Press, 2007.

[25] R. Serrano. bargaining. In S. N. Durlauf and L. E. Blume, editors, *The New Palgrave Dictionary of Economics*. Palgrave Macmillan, Basingstoke, 2008.

[26] M. Thielscher. A general game description language for incomplete information games. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, 2010.

[27] J. von Neumann. On the theory of games of strategy. In A. Tucker and R. Luce, editors, *Contributions to the Theory of Games*, pages 13–42. Princeton University Press, 1959.

[28] D. Zhang and M. Thielscher. A logic for reasoning about game strategies. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15)*, pages 1671–1677, 2015.