Noname manuscript No.
(will be inserted by the editor)

GDL as a Unifying Domain Description Language for Declarative Automated Negotiation

Dave de Jonge · Dongmo Zhang

the date of receipt and acceptance should be inserted later

Abstract Recently, it has been proposed that Game Description Language (GDL) could be used to define negotiation domains. This would open up an entirely new, *declarative*, approach to Automated Negotiations in which a single algorithm could negotiate over any domain, as long as that domain is expressible in GDL. However, until now, the feasibility of this approach has only been demonstrated on a few toy-world problems. Therefore, in this paper we show that GDL is a truly unifying language that can also be used to define more general and more complex negotiation domains. We demonstrate this by showing that some of the most commonly used test-beds in the Automated Negotiations literature, namely Genius and Colored Trails, can be described in GDL. More specifically, we formally prove that the set of possible agreements of any negotiation domain from Genius (either linear or non-linear) can be modeled as a set of strategies over a deterministic extensive-form game. Furthermore, we show that this game can be effectively described in GDL and we show experimentally that, given only this GDL description, we can explore the agreement space efficiently using entirely generic domain-independent algorithms. In addition, we show that the same holds for negotiation domains in the Colored Trails framework. This means that one could indeed implement a single negotiating agent that is capable of negotiating over a broad class of negotiation domains, including Genius and Colored Trails.

Keywords Automated Negotiations · Game Description Language · Non-zero-sum Games · Extensive-form Games · General Game Playing · Monte Carlo Tree Search

Dave de Jonge
IIIA-CSIC
Campus de La UAB, 08193, Bellaterra, Catalonia, Spain
E-mail: davedejonge@iiia.csic.es

Dongmo Zhang
Western Sydney University, School of Computing, Engineering and Mathematics
Locked Bag 1797, Penrith, NSW 2751, Australia
E-mail: d.zhang@westernsydney.edu.au

1 Introduction

One of the long standing goals of Artificial Intelligence has been to develop *general intelligence*. That is, to develop algorithms that are capable of reasoning about virtually any kind of problem, rather than just one pre-specified problem. One important achievement in this direction was the development of Answer Set Programming (ASP) [46], which allows combinatorial problems to be solved using a 'declarative approach'. This means that one only needs to define the problem in a machine-readable language, after which it can be solved automatically by a generic ASP solver.

Another successful application of the declarative approach, is in the field of General Game Playing (GGP) [22], which aims to implement algorithms that are able to play games of which the rules are only known at run-time. This means that such algorithms cannot be implemented to play just one specific game, but instead have to be able to play *any* kind of formally describable game. Just as in the case of ASP, this approach requires a machine-readable language that allows to express a broad class of problem instances (games, in this case). For this purpose the Game Description Language (GDL) [39] was developed.

Recently, de Jonge and Zhang [31] have proposed to apply the declarative approach also to the field of Automated Negotiations, using GDL to specify negotiation domains. The idea is that two agents are playing a non-zero-sum game described in GDL, but they have the opportunity to negotiate binding agreements with each other about which moves each will make. This idea was further developed in [32], which introduced an algorithm for declarative automated negotiation, called Monte Carlo Negotiation Search (MCNS). This algorithm is based on Monte Carlo Tree Search (MCTS), but extended with a negotiation algorithm that allows a player to negotiate with its opponent. It is entirely generic in the sense that it can be applied to any game expressible in GDL and it was tested on three simple games, namely the Iterated Prisoner's Dilemma [1], the Centipede Game [52], and the Dollar Auction [60]. In each of these domains MCNS was able to make (near-)optimal agreements with its opponent.

Although this work did shed light on the feasibility of the declarative approach for Automated Negotiations, it did not demonstrate to what extent it is applicable to more traditional negotiation domains that do not involve games. Therefore, the aim of the current paper is to take this idea a step further and show that this declarative approach can also be applied to two of the most commonly used test-beds in the field of Automated Negotiations, namely Genius and Colored Trails. Although the domains in Genius are not directly related to games, we show that they can nevertheless be modeled as deterministic extensive-form games which can be described efficiently in GDL.

Games are an important research topic in Artificial Intelligence because they provide a controlled environment with clear-cut rules. They can be seen as simplified metaphors for real-world problems. Any multiagent system in which each agent has its own private goals is essentially a game. However, research on game-playing algorithms has mostly focused on zero-sum-games, such as Chess, Poker, and Go. This is striking, because many problems encountered in daily life are more similar to non-zero-sum games. For example, the concept of a market economy is essentially a non-zero-sum game; each participant in the economy has its own personal goals and by exchanging goods and services with one another those participants mutually benefit. Typical examples of games where the outcome is inefficient if players do not cooperate, are the three games mentioned above: the (Iterated) Prisoner's Dilemma, the Centipede Game, and the Dollar Auction. If we adapt these games to allow the players to negotiate and make binding agreements then each individual player can achieve a much better outcome than in the traditional setting without negotiations.

The field of Automated Negotiations, on the other hand, does deal with non-zero-sum situations. In this field one typically assumes that two or more purely self-interested agents have to find mutually acceptable solutions which require cooperation, even though these agents have conflicting goals. Although these agents are not targeting any form of social welfare optimization, each agent is still willing to cooperate if (and typically *only* if) that enables it to achieve a solution that is better than any solution it could realize on its own.

This field, however, traditionally has focused mainly on simpler domains that lack the complexity of extensive-form games. Many papers published on this topic assume the utility of any deal can be known almost instantaneously, and the process of evaluating a proposal is largely abstracted away [13,14,4]. Typically, they assume the value of a deal can be calculated with linear formula, which, we argue, is unrealistic for many real-world negotiation scenarios. We therefore aim to study negotiation domains with much higher complexity, similar to games like Chess and Go for which in most cases it is not feasible to determine the exact value of any action taken.

Many testbeds have been developed to test and implement Automated Negotiation algorithms. Most notably, the Genius framework. This framework contains many negotiation domains, which can be divided into two main types, namely *linear domains*, and non-linear *hypercube domains*. However, Genius still lacks the expressiveness to define more complex negotiation domains with hard constraints between the issues under negotiation. For such complex scenarios, a plethora of alternative test-beds have been developed, such as Colored Trails [23], Diplomacy [12,27], The Negotiating Salemen Problem [28], Wi-Fi Channel Assignment [9], and Settlers of Catan [65,69]. This is unfortunate, because it means that entirely new algorithms need to be defined for every new domain. Algorithms implemented for one domain cannot be reused for another, which makes it hard to compare them.

For these reasons we argue for the use of GDL as a unifying language to define negotiation domains. It allows researchers to define negotiation domains with the complexity of games like Chess or Go, and at the same time allows them to develop generic algorithms that are reusable for different domains, similar to the field of GGP.

Yet another advantage of using games and GDL to model negotiation domains, is that it allows to define domains in which the utility of an agent does not only depend on the agreements it makes, but also on actions and decisions it takes outside the negotiations. We think this is a more realistic model of how negotiations often work in the real world. For example, suppose a customer negotiates with a car salesman to buy a car. If the customer is unmarried and lives in the city then it may be a very good deal to buy a small car which is easy to park and does not consume much fuel. However, if one year later the customer does get married and decides to start a family, that deal suddenly is not so good anymore, as he now needs a larger family car. Interestingly, we see that although the deal itself has not changed, its utility certainly has changed as a consequence of some decisions taken long after the negotiations have finished. A similar thing can happen when we allow agents to negotiate their moves in a game. In that case the agents could, for example, make agreements about only the first few moves they make while retaining the freedom to freely choose their moves in later rounds, which will influence the outcome.

Similarly, an agent's utility may also depend on actions taken by other agents. Imagine, for example, renting a property to open a restaurant in a street with no other restaurants. This might be a good deal at first, until one day multiple other restaurants also open in that same street. This may present the restaurant owner with so much competition that they can no longer afford the rent. Traditional negotiation domains do not take such 'outside actions' into account while they do play a key role in games such as Diplomacy.

In summary, we argue that using GDL to define negotiation domains has the following advantages over traditional domain descriptions:

- It allows for a 'declarative approach' to negotiations, in which all relevant domain knowledge in encoded in the domain description, rather than in the algorithm itself.
- It allows us to use a single language to define a wide class of existing negotiation domains, including Genius and Colored Trails.
- It allows us to write negotiation algorithms that can be applied to all negotiation domains that belong to this class (instead of having to write a new algorithm for every new domain).
- It allows us to define negotiation domains with hard constraints between the issues.
- It allows us to define negotiation domains with the complexity of large extensive-form games such as Chess and Go.
- It allows us to define negotiation domains where the value of a deal does not only depend on the deal itself, but also on actions taken outside the negotiations, either by the agent itself or by its opponents.

In this paper, however, we mainly focus on the second point. Specifically, we aim to answer the following main research question:
Can general negotiation domains be encoded in GDL, in an efficient manner, and such that an entirely generic negotiation algorithm such as MCNS can be applied to them?

In order to answer this question we use Genius and Colored Trails as examples and we present the following results:

- We classify negotiation domains into two broad classes, namely *Cartesian domains* and *Strategic domains*, based on the structure of their agreement spaces.
- We define two notions of equivalence (*isomorphism* and *weak isomorphism*) between negotiation domains.
- We formally prove that every Cartesian domain (which includes all domains in Genius) is weakly isomorphic to some strategic negotiation domain.
- We show that, thanks to this weak isomorphism, any domain from the Genius framework (either linear or non-linear) can be described efficiently in GDL.
- We show that the Colored Trails domains can also be described efficiently in GDL.
- We experimentally show that the agreement spaces of the linear Genius domains and the Colored Trails domains can still be explored efficiently when described in GDL format.

We should remark that in some existing work the process of negotiation *itself* is modeled as an extensive-form game, for example in [48]. In such work the action of proposing, accepting, or rejecting an offer is considered a move in a game. However, that is *not* what we are doing. In this paper, we consider negotiation domains in which agents negotiate which moves they will make in some game that by itself may have nothing to do with negotiations.

2 Related Work

The earliest work on Automated Negotiations mainly focused on proving formal properties of idealized scenarios. A seminal paper of this type is by Nash [45] which showed that under certain axioms the rational outcome of a bilateral negotiation is the solution that maximizes the product of the players' utilities. Afterwards, many papers followed with generalizations or adaptations of these axioms [59].

Later, with the seminal work by Faratin et al. [13, 14], focus shifted more towards heuristic approaches 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. However, Faratin et al. did not take into account that it may be computationally expensive to explore the set of potential agreements and find the right proposal to make. They simply assumed the negotiator could always find it without effort.

Another step forward was made with the development of the Genius framework [38], and subsequently the introduction of the annual Automated Negotiating Agent Competition (ANAC) in 2010 [4], which was run on Genius. Genius is a comprehensive framework that provides all components necessary for implementing a negotiating agent, defining negotiation scenarios, running tournaments and experiments, and analyzing their results. The Genius platform has since become one of the most commonly used test-beds for research on Automated Negotiations. For example, Sanchez-Anguix et al. [53] used it to test a model of intra-team negotiations (negotiations between members of a single team that are to negotiate with another team or opponent). Williams et al. [68] and Chen et al. [6] both used linear domains from Genius to test their respective algorithms to predict their opponents' negotiation strategies. Ilany and Gal [24] used the linear domains from Genius to test a meta-agent that applies machine learning techniques to select the best algorithm for a given negotiation domain, from a set of available algorithms.

Unfortunately, the Genius framework and the first editions of ANAC still made several simplifying assumptions. For example, they assumed there is only a small set of possible agreements, and that the utility functions are linear additive. A next step in the direction of more realistic settings was made with the introduction of the 'hypercube' model [25, 42, 43], which allows for the definition of negotiation domains with non-linear utility functions. Furthermore, these works were some of the first to tackle negotiation domains in which the number of possible deals is too large for exhaustive exploration, so intelligent search algorithms are needed to find good proposals. This hypercube model was later also adopted by Genius and the 2014 edition of ANAC [19] and has become another commonly used model for negotiations research. For example, Aydogan et al. [2], used it to experiment with Machine Learning techniques to learn how to select the best negotiation protocol given a set of characteristics of a negotiation domain.

Although the utility functions in the hypercube model are indeed non-linear over the vector space that represents the set of possible deals, the value of any given deal can still be calculated quickly by solving a linear equation. From a theoretical point of view one can argue that this is not a restriction, because any non-linear function can be modeled in such a way, but in practice the 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). Transforming the given expression of a utility function into the hypercube model would in many cases be a computationally unfeasible task.

Therefore, the idea of complex utility functions was taken a step further in [28], which studied a problem called the Negotiating Salesmen Problem. This problem is an adaptation of the Traveling Salesman Problem that includes negotiations. The utility functions of this negotiation domain are not only non-linear, but also computationally hard to calculate.

An even more complex negotiation scenario is the game of Diplomacy [12]. This is an extensive-form game that involves negotiations before each round. These negotiations are highly complex, because the players' utility functions are not directly defined in terms of the agreements they make, but more indirectly through the moves they make throughout the game. The players negotiate with one another about which moves each will make, which in turn influences the outcome of the game. Determining the effect of an agreement on the player's final utility is a hard problem that involves game theory and constraint satisfaction. Although the earliest work on negotiations in the game of Diplomacy was already conducted as early as 1989 [37,36], new interest in this game as a test-bed for negotiations sparked with the introduction of the DipGame framework [12] and its extension Bandana [30]. Several negotiating agents have been developed on these platforms [11,15,41,30] and from 2017 to 2019 ANAC even hosted a Diplomacy League [27].

Other games that have been studied in combination with negotiations are *Settlers of Catan* [65,69], and *Werewolves* [41]. In all these cases, however, the algorithms were designed to play *one* particular game. The goal of our research, on the other hand, is to design algorithms that negotiate over *any* possible game, similar to the field of General Game Playing. In [33] we studied some of the theoretical properties of games with negotiations.

Another negotiation framework that has been used extensively is the Colored Trails game [20,23]. This framework has traditionally been used mainly to study negotiations between humans and agents. For example, Mell et al. [44] used it for their research on repeated negotiations between humans and virtual agents. Ficici and Pfeffer [16] used Colored Trails to investigate how people reason strategically about others under uncertainty. Peled et al. [50] used it to experiment with an agent that incorporates information revelation decisions into its negotiation strategy and that tries to predict the behavior of its human opponent, and Voynarovskaya et al. [67] even used it as a tool to monitor the mental stability of prospective astronauts who were placed in long-term isolation as a test for a future mission to Mars. On the other hand, de Weerd et al. [10] used Colored Trails in a setting with only automated agents, to investigate the benefits of higher-order theory of mind for such agents.

The field of General Game Playing (GGP) started to draw widespread attention in the AI community after the introduction of Game Description Language (GDL) [39] and the first edition of the annual AAAI GGP competition in 2005 [22] (although earlier work on this topic does exist). Common techniques applied by GGP players are minimax [66], alpha-beta pruning [34] and Monte Carlo Tree Search (MCTS) [35]. Many of the winners of the AAAI GGP competition applied variants of MCTS, such as FluxPlayer [56], Cadia Player [17], Sancho,[1] and Galvanise.[2] MCTS is also the basic algorithm applied by the AlphaGo program that recently defeated the world champion in Go [61].

Pure GDL can only be used to describe deterministic games with full information, but an extension of this language, called GDL-II, was proposed by Thielscher [62] to allow for games with randomness and imperfect information. Next, Thielscher showed that any extensive form game can be described faithfully in GDL-II [63]. A further extension called GDL-III was proposed to include *epistemic* games [64], which have rules that depend on the knowledge of players. Another epistemic variant of GDL, called EGDL, was developed to allow for reasoning about game information and players' epistemic status [26]. Alternative languages to describe video games [55] and card games [18] have also been developed, but these languages are very different from GDL.

The idea that GDL can be used to describe multi-agent environments in general, rather than just classical board games, was already mentioned in [57]. In [8], for example, it was used to represent domain knowledge in mediated dispute resolution. The idea to combine GGP and GDL with Automated Negotiations was put forward by us in [31] and in [32] we presented the MCNS algorithm for such domains. This idea was then further extended by Lv et al. [40], which proposed to use EGDL to model negotiation domains with imperfect information. Another interesting approach to combine GGP and GDL with Automated Ne-

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

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

gotiation was taken by Chitizadeh and Thielscher [7]. In their work they focus on purely cooperative games in which the two agents share the same utility function. They propose an algorithm for the agents to develop a common language at runtime which allows them to propose a mutually beneficial strategy. Note that this is different from our work, since we assume domains in which the two agents have conflicting goals, but cooperation is still better than not cooperating at all. Furthermore, in our case the common language is already fixed in advance, so the agents only need to focus on which agreements to make.

3 Outline

In this paper we consider Automated Negotiation scenarios in which two purely self-interested agents are playing a deterministic non-zero-sum game G. Although these agents are not interested in any form of social welfare optimization, the fact that the game is non-zero-sum means that they may both benefit if they coordinate their actions, so before making their moves they have the opportunity to negotiate with each other about which moves they will or will not play.[3] We will denote this negotiation scenario by $\mathcal{B}(G)$ and we will call negotiation domains of this form *Strategic Negotiation Domains*.

Apart from Strategic negotiation domains, we also consider what we call *Cartesian Negotiation Domains*, which include all domains in Genius. We show that for any Cartesian domain $\mathcal{C}$, we can define a game $G_{\mathcal{C}}$, such that the corresponding strategic domain $\mathcal{B}(G_{\mathcal{C}})$ is essentially the same as $\mathcal{C}$ (we will formalize this, of course). This implies that a negotiation algorithm such as MCNS which is designed for general *strategic* negotiation domains can also be applied to any Cartesian domain, when provided with a GDL description of $G_{\mathcal{C}}$. This is an important result, because the MCNS algorithm was not specifically designed for Genius. It was designed to take any GDL description as its input, which means it is applicable to a much broader class of domains.

The key insight of our work is that any Cartesian product of finite sets can be modeled as a graph (e.g. compare Table 1 and Figure 3). This makes any Cartesian negotiation domain essentially a graph with utility functions, which in turn is, roughly speaking, an extensive-form game. However, the opposite does not hold: not every graph can be modeled as a Cartesian product (e.g. Figure 4), and therefore the class of extensive-form games is more general than the class of Cartesian negotiation domains.

Although everything in this paper can be generalized straightforwardly to multilateral negotiations, we want to keep things simple and therefore always assume there are only two agents, which we denote by α_1 and α_2.

The rest of this paper is organized as follows. In Section 4 we formally define the concept of a Negotiation Domain and the concept of a Cartesian Negotiation Domain. In Section 5 we define the concept of a game, and related concepts such as the minimax value. Then, in Section 6 we link games and negotiation domains together in the form of a Strategic Negotiation Domain. Next, in Section 7 we give a short introduction to GDL, followed by Section 8 in which we explain how Cartesian domains can be defined in GDL, and Section 9 in which we explain how Colored Trails domains can be described in GDL. Next, in Section 10 we present experimental results. In Section 11 we present the formal proof of our claim that any Cartesian Negotiation Domain can be described in GDL. Finally, in Section 12 we discuss a number of issues that we left open, as well as future work.

[3] We assume that when the agents make an agreement this is considered strictly binding, so they are forced to obey it. The question *how* the agreements can be enforced is beyond the scope of our work. We simply assume there is some external mechanism that forces the agents to obey their agreements.

4 Negotiation Domains

In a typical, classical domain for Automated Negotiations two agents α_1 and α_2 are bargaining to agree on some contract. The agents have a fixed amount of time to make proposals to one another according to some negotiation protocol, such as the Alternating Offers protocol [51]. That is, each agent may propose a contract x from some predefined set of possible contracts Ω (known as the *contract space* or *agreement space*) to the other agent and then the other agent may either accept the proposal or reject it and make a counter proposal $y \in \Omega$. The agents continue making proposals to one another until either the deadline has passed, or one of the agents accepts a proposal made by the other.

Each agent α_i has a utility function U_i that assigns to each contract $x \in \Omega$ a utility value $U_i(x) \in \mathbb{R}$. If a contract x proposed by one agent is accepted by the other agent then both agents receive their respective utility values, $U_1(x)$ and $U_2(x)$ corresponding to this contract. On the other hand, if the negotiations fail because no proposal is accepted before the deadline, then each agent α_i will receive a pre-defined utility value which is known as its *reservation value*. We refer to the case that no agreement is made as the *conflict outcome*. A perfectly rational agent would never accept any proposal that yields less utility than its reservation value, because it would be better off by not accepting any proposal at all.[4] Therefore, typically we are only interested in those contracts for which both agents receive a utility value that is at least as high as their reservation value, and for at least one of them this utility should be strictly higher. Such contracts are in the literature called *individually rational*.

A typical example of a negotiation domain would be the case of a car salesman and a client negotiating over the price of the car. In that case the contract space would be the set of all possible prices the customer could possibly pay. However, more complex negotiation scenarios are possible in which the agreement space is multi-dimensional, meaning that they do not only negotiate the price, but also other aspects of the deal, such as the type of engine, the size of the wheels, or any additional features.

Definition 1 A (bilateral) **Negotiation Domain** is a tuple $\langle \Omega, c, U_1, U_2 \rangle$ where:

- Ω is the set of **contracts** (also known as **agreements**, or **deals**).
- c is the **conflict outcome**.
- U_1 and U_2 are two **utility functions** (one for each agent) which are maps from $\mathcal{O}$ to $\mathbb{R}$, where $\mathcal{O} := \Omega \cup \{c\}$

We call $\mathcal{O}$ the set of possible **outcomes** of the negotiation. The utility values $U_i(c)$ for the conflict outcome are called the **reservation values**. Given an outcome $o \in \mathcal{O}$ we call the pair $(U_1(o), U_2(o))$ the **utility vector** of o, and the set of all utility vectors is called the **utility space**.

The utility space of a negotiation domain is typically displayed in a diagram such as in Figure 1.

Definition 2 A contract $x \in \Omega$ is said to be **individually rational** if $U_1(x) \geq U_1(c)$ and $U_2(x) \geq U_2(c)$ both hold, and at least one of those inequalities is strict.

We can distinguish between two types of negotiation domains, based on the structure of the agreement space, namely *Cartesian* domains, for which the agreement space is a finite

[4] In the case the negotiator is *bounded* rational we should say that it would never accept any proposal it *expects* to yield utility less than its reservation value.

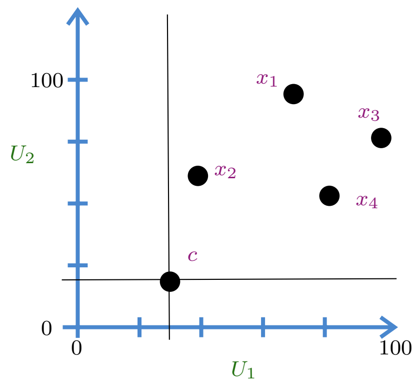

Fig. 1: Utility space diagram of some negotiation domain. Every dot represents the utility vector of an outcome (a contract x_i or the conflict outcome c). The horizontal axis represents the utility of α_1 and the vertical axis represents the utility of α_2. The two black lines represent the reservation values of the two respective agents.

.

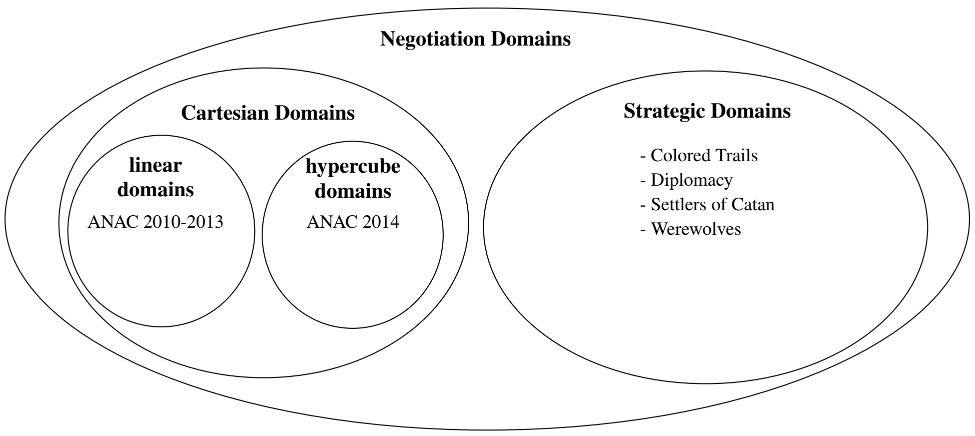

Fig. 2: A classification of the various negotiation domains studied in the literature, based on the structure of their agreement spaces.

Cartesian product of finite sets, and *Strategic* domains, for which the agreement space is represented as a graph, or, more precisely, as an extensive-form game. This classification is displayed in Figure 2.

Definition 3 A negotiation domain $\langle \Omega, c, U_1, U_2 \rangle$ is called a **Cartesian negotiation domain** if its contract space Ω is the Cartesian product of a finite number of finite sets $\Omega = I_1 \times I_2 \cdots \times I_n$, which are called the **issues** of the domain.

Typical examples of Cartesian domains are the domains in Genius. The idea is that the agents need to agree on a contract which is composed of several components whose values can be freely combined. For example, when buying a computer the issues under negotiation could be the type of CPU, the amount of RAM and the size of the hard disk. If we assume that each of these can be freely combined with one another then the entire agreement space is indeed the Cartesian product of the issues.

Furthermore, within the Cartesian domains we can distinguish between domains with linear and with non-linear utility functions. In the case of linear domains, each agent has a preference relation over each issue, which is completely independent of the other issues.

Definition 4 A Cartesian negotiation domain is called a **linear negotiation domain** if its utility functions U_i are given by:[5]

$$U_i(a_1, a_2, \dots a_n) = \sum_{j=1}^{n} d_{i,j}(a_j) \tag{1}$$

where each $a_j \in I_j$ and each $d_{i,j}$ is a map from I_j to $\mathbb{R}$.

Note that Eq. (1) does not completely define the utility functions, because it does not specify the reservation values. So, to completely define a linear negotiation domain one must also specify the reservation values $U_i(c)$ separately.

An example of a linear negotiation domain is given in Table 1. This domain represents the case where a customer with a fixed budget of, say, 500 euro wants to buy a phone and negotiates with a salesperson about which phone can be purchased for that price. In this example, there are three 'issues' under negotiation, namely the brand of the phone, the color of the phone and the size of its memory. The first issue has three possible values: $I_1 = \{\text{Samsung}, \text{Apple}, \text{Huawei}\}$, while the other two issues have two possible values each. $I_2 = \{\text{Black}, \text{Silver}\}$, $I_3 = \{16\text{ GB}, 32\text{ GB}\}$. This means there are 3x2x2 possible deals that the negotiators can choose from. An example of a deal would be $x = (\text{Samsung}, \text{Black}, 16\text{ GB})$, which yields a utility of $U_1(x) = 0+20+18 = 38$ to the buyer and $U_2(x) = 40+30+10 = 80$ to the seller. Note that in this particular example we assume that the customer wants to buy the best possible phone within its budget, so the price is not part of the negotiations. In general, however, it is perfectly possible that the price is also one of the issues under negotiation.

The non-linear Cartesian domains in the Genius framework are so-called *hypercube* domains. In the hypercube domains the agents' preferences over the issues no longer have to be independent. For example, if a customer chooses to buy an Apple phone he might prefer it to be black, while if he chooses to buy a Samsung phone, he prefers it to be silver. In that case, the issue 'color' depends on the issue 'brand'. Hypercube domains were first introduced in [25]. Here, we define them as follows.

Definition 5 Let $\Omega = I_1 \times I_2 \times \dots I_n$ be a Cartesian product of finite sets. Then a **hypercube** (or **hyperrectangle**) $h \subseteq \Omega$ of this product is another Cartesian product $J_1 \times J_2 \times \dots J_n$ where each J_i is a non-empty subset of I_i (possibly equal to I_i).

For any hypercube h we define its associated Kronecker delta function $\delta_h : \Omega \to \{0, 1\}$ as follows:

$$\delta_h(\vec{a}) = \begin{cases} 1 & \text{if } \vec{a} \in h \\ 0 & \text{if } \vec{a} \notin h \end{cases}$$

[5] In the literature the utility functions are usually defined as *weighted* sums. However, we can omit the weights without loss of generality, because they can be absorbed into the $d_{i,j}$.

Issues	Values	Utility Buyer	Utility Seller
Brand	Samsung	0	40
	Apple	25	59
	Huawei	39	0
Color	Black	20	30
	Silver	30	30
Memory	16 GB	18	10
	32 GB	30	0
Reservation values:		10	35

Table 1: The Mobile Phone Domain. This is an example of a linear negotiation domain with 3 issues, namely 'Brand', 'Color', and 'Memory'. The first issue has 3 possible values, while the other two have 2 possible values. Therefore, the agreement space contains a total of $3 \times 2 \times 2 = 12$ possible deals. If the buyer buys a silver Huawei phone with 32 GB memory he would receive a total utility of 39 + 30 + 30 = 99, which happens to be the best possible deal. However, for the seller that would result in 0 + 30 + 0 = 30 utility, which is the lowest possible. Moreover, this value is lower than the seller's reservation value, which is 35. Therefore, for the seller this deal would be even worse than not making any deal at all.

Definition 6 A Cartesian negotiation domain $\mathcal{C}$ is called a **hypercube domain** if its utility functions U_i are given by:

$$U_i(\vec{a}) = \sum_{h \in H} w_h \cdot \delta_h(\vec{a})$$

where $\vec{a} \in \Omega$, H is a set of hypercubes of Ω, and for every $h \in H$ we have $w_h \in \mathbb{R}$. A pair (h, w_h) is called a **constraint**.

Although hypercube domains allow for soft constraints between the issues, they still do not allow hard constraints. If there are hard constraints, the agreement space can no longer be modeled as a Cartesian product of the issues, because some of the tuples would be invalid. For example, suppose that the Samsung phone is only available in black and silver, while the Apple phone is available in also in black, silver, and blue. Such a domain cannot be represented as a Cartesian product. For such domains, we need a more complex structure, such as a graph (see Figure 4). This is captured by the notion of a Strategic Negotiation Domain, which we define in Section 6.

Regarding to our classification of Figure 2 we should remark that we are by no means claiming this classification is exhaustive. It may certainly be possible to define negotiation domains that are neither Cartesian, nor Strategic. Similarly, it may be possible to define Cartesian domains that are neither linear, nor hypercube. However, we do think that these classes represent the majority of the negotiation domains studied in the literature.

Another important remark is that the classification of a given negotiation domain may depend on how exactly it is modeled. For example, if some domain A is a Cartesian negotiation domain and some other domain B is a Strategic negotiation domain it means they have very different mathematical structures. Nevertheless, it is perfectly possible that they both represent exactly the same real-world negotiation scenario. In fact, one of the main results of this paper is that we show that every Cartesian domain can just as well be modeled as a Strategic domain.

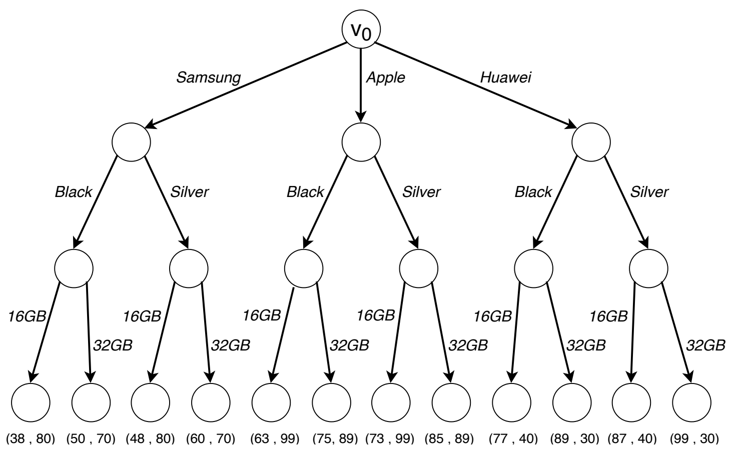

Fig. 3: Every Cartesian product can be modeled as a graph, and therefore every Cartesian Negotiation Domain can be modeled as a game. This figure depicts the game $G'_{\mathcal{C}}$ corresponding to the linear negotiation domain $\mathcal{C}$ displayed in Table 1.

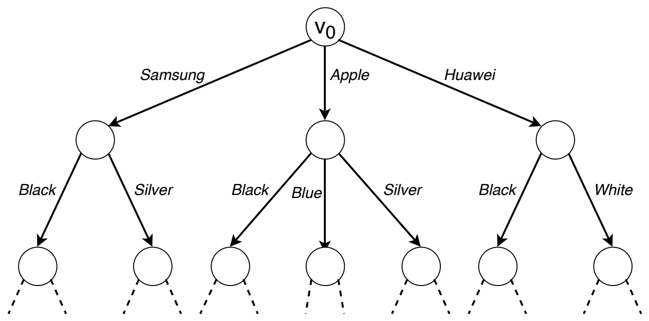

Fig. 4: Not every graph can be modeled as a Cartesian product. For example this graph does not represent a Cartesian product, because the values of the issue 'color' depend on the brand.

5 Extensive-Form Games

Before we can define Strategic Negotiation Domains we first need to define the concept of a *game*. Technically speaking, the games we define here are *deterministic two-player extensive-form turn-taking games with perfect information and finite horizon* [49]. However, we will simply refer to them as 'games'. Typical examples of such games are chess, checkers, and go. Again, we let α_1 and α_2 denote the two agents playing the game.

Definition 7 A **game** is a tuple $\langle V, E, v_0, T, p, A, m, u_1, u_2 \rangle$, where:

- $\langle V, E \rangle$ is a finite directed acyclic graph with vertices V (a.k.a. **states**) and edges E.
- $v_0 \in V$ is the **initial state**.

- $T \subset V$ is the set of **terminal** states. That is, the set of states that have no outgoing edges.[6]
- p is a vertex-label function $p : V \setminus T \to \{1, 2\}$, known as the **active player function**. For any non-terminal state v the agent $\alpha_{p(v)}$ is called the **active player** of that state.
- A is a set of **actions** (or **moves**).
- m is an edge-label function $m : E \to A$, known as the **move function**, that assigns an action to every edge. It must satisfy the constraint that for any vertex v, its outgoing edges must all be labeled with a different move:[7]

$$\forall (v, w), (v, w') \in E \quad : \quad w \neq w' \quad \to \quad m(v, w) \neq m(v, w')$$

- u_1 and u_2 are the players' respective **utility functions** $u_i : T \to \mathbb{R}$.

If the game is a classical board game like chess, then each state $v \in V$ represents a particular configuration of the pieces on the board, and the *active player* is the agent whose turn it is to make a move. If v has an outgoing edge (v, w) labeled with an action a it means that if the active player chooses to play a then the next state resulting from that move will be w. The terminal states represent those configurations in which the game is finished (in chess that would be a checkmate or a stalemate).

It is important to understand that a game according to this definition does not necessarily have to be a *board* game. In fact, in this paper we are not so much interested in 'games' in the informal sense of the word, but rather we see games as a purely abstract mathematical concept. A game is just a graph decorated with utility functions, an active-player function and a move-function.

As an example, we now define the Turn-Taking Prisoner's Dilemma (TTPD), which will turn out to be important later on in this paper. In the TTPD player α_1 moves first, and has two options: either to play 'cooperate' or to play 'defect'. Next, player α_2 moves, and has the same two options. After this the game is over. Just as in the traditional Prisoner's Dilemma the utility functions are defined such that if the players are rational they will both play 'defect' even though they would both be better off if they both played 'cooperate'. Its formal definition is as follows, but it may be easier to understand from Figure 5.

Definition 8 The **turn-taking prisoner's dilemma** (TTPD) is the game $\langle V, E, v_0, p, A, m, u_1, u_2 \rangle$ defined as follows

- $V = \{v_0, v_d, v_c, t_{dd}, t_{dc}, t_{cd}, t_{cc}\}$
- $E = \{(v_0, v_d), (v_d, t_{dd}), (v_d, t_{dc}), (v_0, v_c), (v_c, t_{cd}), (v_c, t_{cc})\}$.
- $T = \{t_{dd}, t_{dc}, t_{cd}, t_{cc}\}$.
- $p(v_0) = 1$, and $p(v_d) = p(v_c) = 2$.
- $A = \{\mathit{defect}, \mathit{cooperate}\}$
- $m(v_0, v_d) = m(v_d, t_{dd}) = m(v_c, t_{cd}) = \mathit{defect}$
  $m(v_0, v_c) = m(v_c, t_{cc}) = m(v_d, t_{dc}) = \mathit{cooperate}$
- $u_1(t_{dd}) = r_1, \quad u_2(t_{dd}) = r_2$
  $u_1(t_{dc}) = 100, \quad u_2(t_{dc}) = 0$
  $u_1(t_{cd}) = 0, \quad u_2(t_{cd}) = 100$
  $u_1(t_{cc}) = q_1, \quad u_2(t_{cd}) = q_2$
  where r_1, r_2, q_1 and q_2 can be arbitrary real numbers, provided that they satisfy
  $\forall i \in \{1, 2\} : 0 < r_i < q_i < 100$.

[6] Note that strictly speaking it is not necessary to include T in this tuple, because T is already implied in the definition of the graph $\langle V, E \rangle$. However, we prefer to include it, for clarity.

[7] Note that we use the notation $m(v, w)$ as a shorthand for $m((v, w))$.

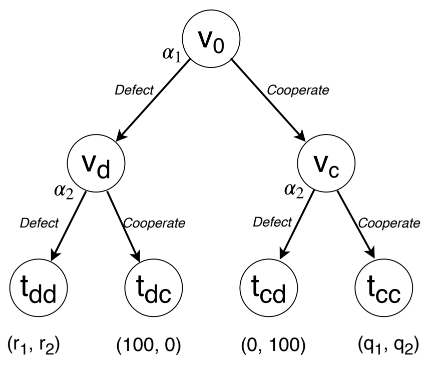

Fig. 5: The turn-taking prisoner's dilemma. The terminal states are indicated with a t. The values r_1, r_2, q_1 and q_2 can be any real numbers that satisfy $0 < r_i < q_i < 100$.

If agents are to negotiate which moves they will play, they need to be able to determine the utility they will obtain from such deals. More generally, we need to be able to answer the following question: "*if the game is in a state v, which utility values can the players expect to achieve at the end of the game, if they both play perfectly rationally, and without making any (further) agreements?*".

In order to answer this question we need to distinguish between two situations: the case that each player only knows its own utility function, and the case that each player also knows their opponent's utility function. For the first case we use the well-known concept of the *minimax* value, while for the second case we use a concept that we call the *pessimistic maximax* value.

It is well-known that if a game is a zero-sum game with full information, then the minimax values of a state v are indeed exactly the utility values the players receive if they play the Subgame Perfect Equilibrium (SPE) from that moment onward [49]. In this paper, however, we are dealing with non-zero-sum games. Nevertheless, we can still use the minimax values, but their interpretation is slightly different. In our case, the minimax values are the *minimum* values the two players can guarantee for themselves, assuming that each of them does not know the other player's utility [54]. In Fig. 6, for example, we see that the minimax values of the TTPD are r_1 for player α_1 and r_2 for player α_2, and that each player can calculate its own minimax value, without knowing the other player's utility.

Definition 9 Let G be a game. Then, for each state v and agent α_i its **minimax value** $\tilde{u}_i(v)$ is defined as:

$$\tilde{u}_i(v) := \begin{cases} u_i(v) & \text{if } v \in T \quad (v \text{ is a terminal state}) \\ \tilde{u}_i(v^{max,i}) & \text{if } p(v) = i \quad (\text{it is } \alpha_i\text{'s turn}) \\ \tilde{u}_i(v^{min,i}) & \text{if } p(v) \neq i \quad (\text{it is the opponent's turn}) \end{cases}$$

where:

$$v^{max,i} \in \arg\max_{v'}\{\tilde{u}_i(v') \mid (v,v') \in E\} \quad (\text{any child of } v \text{ that maximizes } \alpha_i\text{'s utility}).$$

$$v^{min,i} \in \arg\min_{v'}\{\tilde{u}_i(v') \mid (v,v') \in E\} \quad (\text{any child of } v \text{ that minimizes } \alpha_i\text{'s utility}).$$

The **minimax values** $\tilde{u}_i(G)$ **of a game** are defined as the minimax values of its initial state, i.e. $\tilde{u}_i(G) := \tilde{u}_i(v_0)$.

Note that since a player can be indifferent between multiple actions, the $\arg\max$ and $\arg\min$ operators each return a *set* of child states with the same minimax values. If such a set contains more than one element it does not matter which of these we choose to be $v^{max,i}$ or $v^{min,i}$, since we only care about its minimax value anyway.

In the case that players do know their opponent's utility (or they can reasonably estimate it), then the minimax assumption is too strong, because we can assume the opponent will pick the actions that maximize its own utility rather than those that minimize our agent's utility. The values obtained under that assumption are sometimes called the *maximax* values. Unfortunately, the maximax values are not always well-defined, because if the active player is indifferent between several optimal actions, while the other player is not, then it matters which action the active player will choose. If that happens we still need to make the 'pessimistic' assumption that, among those actions that maximize our opponent's utility, he will choose the one that minimizes our own utility. We call the values we obtain under this assumption the *pessimistic maximax values*. It is not hard to see that indeed they represent the minimum utility that rational players can guarantee themselves without any form of coordination.

Definition 10 Let G be a game, then for any non-terminal state v and player α_i, let $V^{max,i}$ denote the set of children of v that maximize α_i's utility:

$$V^{max,i} := \arg\max_{v'} \{\tilde{u}_i(v') \mid (v,v') \in E\}$$

Furthermore, let $v^{pess-max,i}$ denote any state among $V^{max,i}$ that minimizes α_j's utility (with $i \neq j$):

$$v^{pess-max,i} \in \arg\min_{v'} \{\tilde{u}_j(v') \mid v' \in V^{max,i}\}$$

Then the **pessimistic maximax value** $\tilde{u}_i(v)$ for state v and player α_i is defined as:

$$\tilde{u}_i(v) := \begin{cases} u_i(v) & \text{if } v \in T \quad (v \text{ is a terminal state}) \\ \tilde{u}_i(v^{pess-max,1}) & \text{if } p(v) = 1 \quad (\text{it is } \alpha_1\text{'s turn}) \\ \tilde{u}_i(v^{pess-max,2}) & \text{if } p(v) = 2 \quad (\text{it is } \alpha_2\text{'s turn}) \end{cases}$$

The **pessimistic maximax values** $\tilde{u}_i(G)$ **of a game** are defined as the pessimistic maximax values of its initial state, i.e. $\tilde{u}_i(G) := \tilde{u}_i(v_0)$.

In the rest of this paper we often just say 'maximax' when we mean 'pessimistic maximax'

Note that we are using the same notation $\tilde{u}_i$ both for the minimax values and for the pessimistic maximax values, because it should be clear from context which one of the two we mean (pessimistic maximax if the opponent's utility is known, minimax if the opponent's utility is unkown).

For very large and complex games like chess and Go it is often computationally unfeasible to determine the minimax or maximax values exactly. In such cases, however, one can apply approximation algorithms such as Monte Carlo Tree Search.

Finally, we should stress that we are not saying that we are assuming the players are really playing according to a minimax or maximax strategy. We are only saying that, *in order to calculate the players' minimum achievable utility, we have to reason as if both players were using such strategies*.

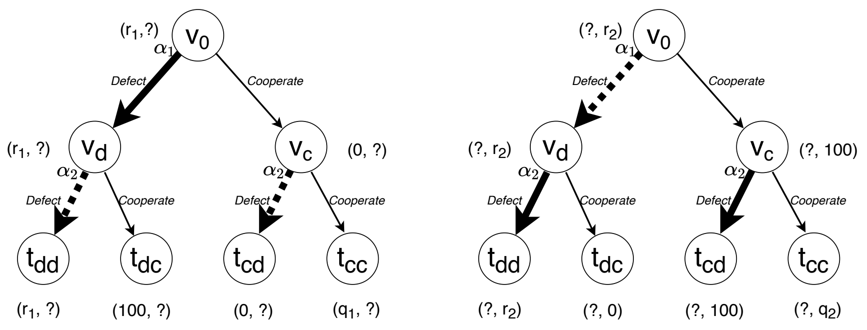

Fig. 6: The turn-taking prisoner's dilemma from the point of view of α_1 (left) and α_2 (right) respectively, if each agent only knows its own utility. The values next to the non-terminal states represent the minimax values calculated by each agent. A thick solid arrow represents a move the agent would choose to maximize its own utility. A thick dashed arrow represents a move the agent's opponent would choose to minimize the agent's utility.

6 Strategic Negotiation Domains

In this section we link the concepts of a Game and a Negotiation Domain together. For a game G we define a *Strategic Negotiation Domain* over G to be a negotiation domain in which the agents negotiate which moves they will make in G. Formally, we say the agents negotiate about which *restriction* of G they will play. A restriction R of G is a game that is obtained from G by removing a number of edges while everything else remains the same. An example of a restriction of the turn-taking prisoner's dilemma is shown in Figure 7.

Definition 11 A game $R = \langle V^R, E^R, v_0^R, T^R, p^R, A^R, m^R, u_1^R, u_2^R \rangle$ is a **restriction**[8] of a game $G = \langle V, E, v_0, T, p, A, m, u_1, u_2 \rangle$ if all of the following hold:

- $\langle V^R, E^R \rangle$ is a subgraph of $\langle V, E \rangle$
- $v_0^R = v_0$
- $T^R \subseteq T$
- $p^R(v) = p(v)$ for all $v \in V^R \setminus T^R$
- $A^R = A$
- $m^R(v) = m(v)$ for all $v \in V^R$
- $u_i^R(t) = u_i(t)$, for $t \in T^R$.

We will use $Rest(G)$ to denote the set of all restrictions of G.

If the agents agree to commit themselves to a restriction R then the utility values they can expect to obtain are the minimax or maximax values $\tilde{u}_i(R)$ of R. In non-zero-sum games such as the TTPD there may be restrictions for which $\tilde{u}_i(R) > \tilde{u}_i(G)$ for both players, meaning that both players would benefit from an agreement to commit themselves to R.

Definition 12 Let G be an unambiguous game. Then a **Strategic Negotiation Domain** over G is a negotiation domain $\langle \Omega, c, U_1, U_2 \rangle$ for which the following holds:

- $\Omega \subseteq Rest(G)$,

[8] One might be tempted to use the term *subgame* instead of *restriction*, but the term subgame already has a different meaning in Game Theory. In [33] we used the concept of a *strategy*, instead of a restriction. Although strategies and restrictions are defined differently, they represent essentially the same thing.

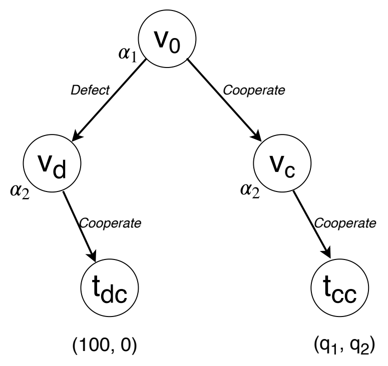

Fig. 7: An example of a restriction of the turn-taking prisoner's dilemma. This particular example corresponds to the agreement that α_2 will not play 'defect'. However, many other restrictions of the TTPD are possible.

– $c = G$
– $U_i(R) = \tilde{u}_i(R)$ for each restriction $R \in \Omega$
– $U_i(c) = \tilde{u}_i(G)$

The statement $\Omega \subseteq Rest(G)$ means that in a strategic negotiation domain over G the agents are proposing restrictions of G to one another. If they agree on a restriction R it means that they are from that moment onwards only allowed to play moves that are in R. This means that, if they are perfectly rational, the utility values they will receive at the end will be exactly the minimax values $\tilde{u}_i(R)$ of that restriction (or maximax if the opponent's utility is known), which is expressed as $U_i(R) = \tilde{u}_i(R)$. The statement $c = G$ expresses the idea that if they do not come to any agreement, then they are not restricted at all, so they can freely choose any move from the game G. This implies that in that case they will each end up with the minimax or maximax value of G, which is expressed by the statement $U_i(c) = \tilde{u}_i(G)$.

In other words, *if N is a strategic negotiation domain over G, and the players do not know each others' utility functions, then the reservation values of N are exactly the minimax values of G. If the players do know each others' utility functions, then the reservation values of N are exactly the pessimistic maximax values of G*

A good example where the notion of a Strategic Negotiation Domain naturally arises, is the game of Diplomacy. In Diplomacy, each round of the game is preceded by a negotiation stage in which the players negotiate about which moves they will or will not make. If G denotes Diplomacy without negotiations (known as *no-press Diplomacy*) then each negotiation stage of the full Diplomacy game can be formalized as the Strategic Negotiation Domain over G (see also [30]). Another example is Colored Trails.

In this paper we are mainly interested in a specific type of restriction, which we call a *branch*. A branch of a game G is a restriction for which the underlying graph is just a path from the initial state to some terminal state. An example of a branch of the TTPD is given in Figure 8.

Definition 13 A **branch** B of a game G is a restriction of G for which every non-terminal state has exactly one outgoing edge. The set of all branches of G is denoted $Branch(G)$.

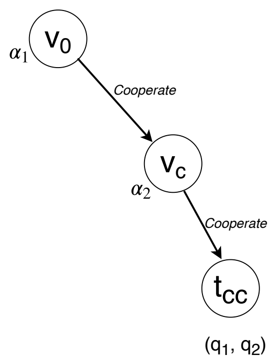

Fig. 8: An example of a branch of the Turn-Taking Prisoner's Dilemma. A branch is a restriction in which each player always has exactly one possible action, so the underlying graph is just a path from the initial state to a terminal state. In this particular example both players can only play 'cooperate'.

Definition 14 For any game G its corresponding **Branch Negotiation Domain** $\mathcal{B}(G)$ is the strategic negotiation domain over G for which the possible agreements are exactly the branches of G. That is: $\Omega = Branch(G)$.

Example As an example, suppose that G is the Turn-Taking Prisoners Dilemma (Def. 8, Fig. 5). The TTPD has four branches, respectively labeled with (*defect*, *defect*), (*defect*, *cooperate*), (*cooperate*, *defect*), and (*cooperate*, *cooperate*). Therefore, the two agents in the negotiation domain $\mathcal{B}(G)$ can propose these four branches to one another as the possible deals. In case they do not come to an agreement the players have no reason to deviate from the subgame perfect equilibrium, so they will each play *defect*, and the game ends in state t_{dd}, yielding the two players a utility of r_1 and r_2 respectively. On the other hand, if they agree to play the branch labeled with (*cooperate*, *cooperate*) the game will end up at state t_{cc}, yielding the two players a utility of q_1 and q_2 respectively. This is for both players a better outcome, since the definition of the TTPD states that $r_i < q_i$. Of course, this only works if we assume that any agreement made is *binding*. So even though it would still be rational for both players to play *defect* they are simply no longer allowed to do so.

The reason that we mainly focus on Branch Negotiation Domains, as opposed to Strategic Negotiation Domains in general, is that we show they can be used to represent domains from Genius. In Genius, the only choice the agents have, is which contract to agree upon. Indeed, in a Branch Negotiation Domain, after the agents have agreed to play a certain branch B, they also have no further choices to make. The outcome of the game is completely fixed. On the other hand, if we allowed more general types of restrictions, the agents would be allowed to make agreements that do not completely fix all their moves, and leave some room for the agents to make their own choices. This is more common in Diplomacy and Colored Trails.

The main claim in this paper is that every Cartesian Negotiation Domain $\mathcal{C}$ can be modeled as the Branch Negotiation Domain $\mathcal{B}(G_{\mathcal{C}})$ over some game $G_{\mathcal{C}}$. To make this formal, we define the notions of *isomorphism* and *weak isomorphism* between negotiation domains. However, since the formal definitions require a number of technical preliminaries, we prefer for now to only give informal definitions and defer the formal definitions until Section 11.

Informally, two negotiation domains are isomorphic if there exists a one-to-one mapping between all the possible outcomes of both domains that preserves the utility values (up to a linear transformation). For two negotiation domains to be *weakly* isomorphic it is enough if there is such a utility-preserving one-to-one mapping between only the *individually rational* outcomes. We can then state our main claim as follows.

Theorem 1 *For any Cartesian Negotiation Domain $\mathcal{C}$ there exists a game $G_{\mathcal{C}}$ such that $\mathcal{B}(G_{\mathcal{C}})$ is weakly isomorphic to $\mathcal{C}$.*

This theorem implies that the MCNS algorithm should be able to negotiate over any domain $\mathcal{C}$ from the Genius framework if we provide it with a GDL description of $G_{\mathcal{C}}$. This is an important result, because the MCNS algorithm does not 'know' anything about Genius, and has previously only been applied to entirely different domains.

Again, we do not want to go into technical details yet, so the proof is given in Section 11. The idea, however, can be demonstrated easily with an example. Say that $\mathcal{C}$ is the linear domain given in Table 1. Then it is easy to see that we can construct a game $G'_{\mathcal{C}}$ as in Figure 3 for which each branch corresponds to exactly one contract in the negotiation domain. Furthermore, the utility values of these branches are exactly the utility values of the corresponding contracts in $\mathcal{C}$.

However, there is one problem with this idea. The problem is that the reservation values of $\mathcal{C}$ are not necessarily equal to the reservation values of $\mathcal{B}(G'_{\mathcal{C}})$. After all, the reservation values of $\mathcal{C}$ can be any arbitrary pair of values, independent of the values of the contracts. On the other hand, the reservation values of $\mathcal{B}(G'_{\mathcal{C}})$ cannot be chosen arbitrarily, because they are equal to the minimax values of the underlying game $G'_{\mathcal{C}}$, which are fixed by the structure of the graph and the utility values of the terminal states.

For this reason we need to apply a trick that allows us to adjust the minimax values. This trick is visualized in Figure 9. The idea is that we define a game $G_{\mathcal{C}}$ that starts as a TTPD, and only if both players choose the *cooperate* move the game continues as the game $G'_{\mathcal{C}}$ we defined previously. The point is that if the players do not make any agreements, they have no reason to play *cooperate*, and just as in the TTPD, the game will end with both players playing *defect*, and they will respectively obtain the utility values r_1 and r_2 from Def. 8, which can be chosen arbitrarily. Therefore, we can choose to set them equal to the reservation values of $\mathcal{C}$.

Although we have now solved the problem of setting the reservation values correctly, by doing so we did introduce a new problem: we have also added three more branches that do not correspond to any contract in $\mathcal{C}$, namely the ones ending in t_{dd}, t_{dc} and t_{cd}, which means that $\mathcal{B}(G_{\mathcal{C}})$ is not isomorphic to $\mathcal{C}$. Luckily, however, this is not a big problem, because none of these is individually rational (t_{dd} yields exactly the reservation values for both agents, while the other two each yield value 0 for one of the two agents). So at least the two domains are still *weakly* isomorphic. We therefore argue that the two negotiation domains are still essentially the same, because rational agents would never agree on an irrational deal anyway (and even for *bounded* rational agents it is still easy to see these deals are not rational, as we show later in our experiments).

7 Game Description Language

In this section we give a short introduction to GDL. For more details we refer to [39].

GDL is a first-order logical language that was designed to describe deterministic extensive-form games. In principle, it can describe any game G defined according to Definition 7.

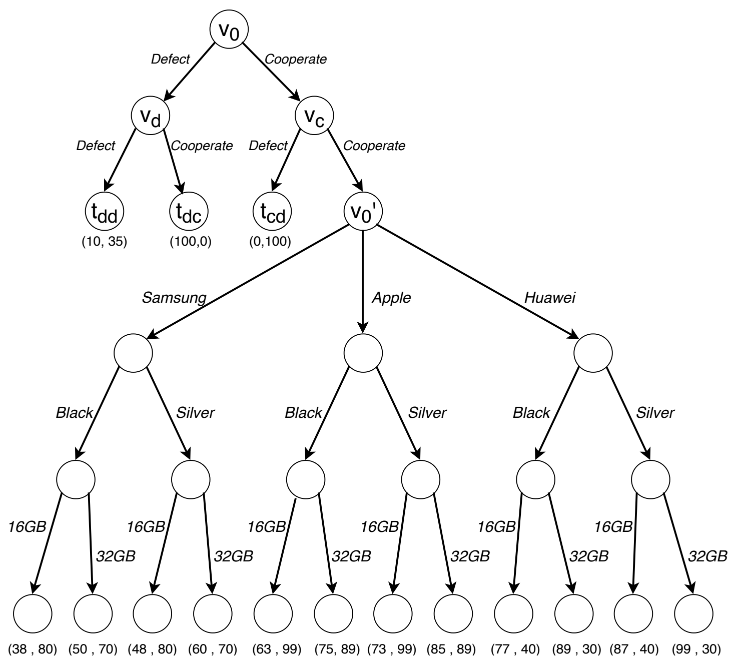

Fig. 9: The game $G_{\mathcal{C}}$ corresponding to the linear negotiation domain $\mathcal{C}$ displayed in Table 1. If the players do not make any agreements about their moves and they play rationally then they will both play *Defect* and end up with 10 and 35 utility points respectively, which are exactly the reservation values of $\mathcal{C}$.

Just as in classical first-order logic (FOL), statements in GDL are composed of constants, variables, function symbols and relation symbols, which can be combined via conjunction, implication and negation. Variables are denoted with a question mark, e.g. `?x`. Just as in FOL, variables and constants are called *terms* and any function symbol acting on a term yields a new term. In classical FOL a function symbol f acting on a constant c or a variable x is denoted as $f(c)$ or $f(x)$. In GDL, however, these terms would be denoted as `(f c)` and `(f ?x)` respectively. Similarly, if r is a relation symbol and τ is a term, then r applied to τ is denoted as `(r` τ`)`, which is called an *atom*. An atom can be negated with the keyword `not`, for example: `(not (r` τ`))`. An atom or a negated atom is called a *literal*. One major difference between FOL and GDL is that negation in GDL is interpreted as *negation-by-failure*. This means that a negative literal `(not p)` is considered true if and only if there is no rule from which one can derive the truth of p.

Literals can be put together to form *rules*, which are of the form:

```
(<= h s1 s2 ... sn)
```

where each s_i is a positive or negative literal, and h is a positive literal. If the rule does not contain any variables it can be roughly interpreted as the FOL statement $s_1 \wedge s_2 \wedge \ldots s_n \rightarrow h$. If it does contain variables, say `?x`, `?y` and `?z` then it can be interpreted as the universally quantified FOL formula $\forall x, y, z\big(s_1 \wedge s_2 \wedge \ldots s_n \rightarrow h\big)$. The atom h is called the *head* of the rule and the s_i's are called the *subgoals* of the rule. The list of subgoals is called the *body* of the rule. The body of a rule may be empty, so it has the form `(<= h)`, meaning that `h` is always true. Such a rule is called a *fact*, and may also be denoted simply as `h`.

The idea behind GDL is that any state v of a game is represented as a set of atoms of the form `(true` τ`)`, where τ can be any ground term (i.e. a term without variables). 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:

$$v = \{\ \texttt{(true (cell 2 2 X))}\ ,\ \texttt{(true (cell 1 1 O))}\ \}$$

where `X` and `O` are user-defined constants and `cell` is a user-defined function. On the other hand, the relation `true` is a keyword of GDL. Other important keywords of GDL are the relation symbols `role`, `goal`, `next`, `init`, `legal`, and `terminal`.

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

$$\texttt{(<= (next q) (true p) (does } \alpha_i \texttt{ a))}$$

it means that if the game is in any state v for which `(true p)` $\in v$ and player α_i plays action a then in the next round the game will be in a state v' for which `(true q)` $\in v'$ holds. Similarly: `(<= terminal (true p))` means that any state v for which $true(p) \in v$ holds is a terminal state. The fact `(<= (init p))` means that for the initial state v_0 we have `(true p)`$\in v_0$. The rule `(<= (legal` α_1 `a) (true p))` means that for any state in which `(true p)`$\in v$ holds it is legal for α_1 to play the move a. The rule `(<= (goal` α_1 `100) (true p))` means that in any state v for which `(true p)`$\in v$ holds α_1 receives a utility value of 100 (which is only meaningful if v is a terminal state). Finally, we mention that in GDL anything following a semi-colon is a comment.

In order to parse a game from a GDL description and apply game-playing algorithms like MCTS one needs to implement a so-called *State Machine*. This is a data structure that is initialized with a set of GDL rules and that allows one to query for any given state whether it is terminal or not, which are the legal actions in that state (if non-terminal), which are the players' utility values (if terminal), and what would be the next state if the active player plays some given legal action. Such a state machine can be implemented in many ways, but a common approach is to implement it based on a so-called PropNet [58].

8 Cartesian Domains in GDL

In Section 6 we argued that a generic algorithm such as MCNS should be able to negotiate over Genius domains, as long as we provide it with a GDL description of the game $G_{\mathcal{C}}$. The question, however, is whether it is actually possible to efficiently generate such a GDL description, given a description of $\mathcal{C}$ in Genius format. After all, the size of the game $G_{\mathcal{C}}$ displayed in Figure 9 is exponential in the number of issues. In this section we show that this is not a problem, because the GDL description of $G_{\mathcal{C}}$ is still linear in size. Furthermore, we show that the utility functions of the linear and the hypercube domains can both be described in GDL.

8.1 Linear Domains

The graph underlying the game $G_{\mathcal{C}}$ of Figure 9 has a very simple structure, and its active player function and move function also follow simple patterns, so it is not difficult to implement these structures in GDL. The main challenge is to implement the utility functions,

```
;; If one player defected and the other cooperated, the defecting player gets 100 points
;; and the cooperating one gets 0 points.
(<= (goal ?d 100)(true (defected ?d))(not(true (defected ?c)))(role ?c))
(<= (goal ?c   0)(true (defected ?d))(not(true (defected ?c)))(role ?d))

;; If both players defected they receive their respective reservation values.
(<= (goal ?d ?rv)(true (defected player1))(true (defected player2))(resval ?d ?rv))

;; Here, (sum_util ?p ?k ?u) represents the fact that the sum of the utility from the first ?k issues equals ?u
(<= (goal ?p ?g)(sum_util ?p ?n ?g)(numIssues ?n)(not(true (defected player1)))(not(true (defected player2))))
(<= (sum_util ?p 1 ?u)(util_current_value ?p 1 ?u))
(<= (sum_util ?i ?k ?sn)(succ ?m ?k)(sum_util ?i ?m ?sm)(util_current_value ?i ?k ?u)(plus ?sm ?u ?sn))

;;the utility ?u that player ?p obtains when the issue with index ?j
(<=(util_current_value ?p ?j ?u)(true (currentContract ?j ?v))(util ?p ?j ?v ?u))
```

Fig. 10: GDL rules for linear utility functions.

because GDL only has very limited support for arithmetics. Nevertheless, we have managed to implement linear utility functions of Equation (1) in GDL, as displayed in Figure 10.

The first three rules define the utility functions of the TTPD at the start of the game. The predicate `(sum_util ?i ?k ?sn)` represents the partial sum $sn_{i,k} = \sum_{j=1}^{k} d_{i,j}(a_j)$ consisting of the first k terms of Equation (1). More precisely, the fifth rule defines $sn_{i,1} = d_{i,1}(a_1)$, and the sixth rule defines the recursive relation $sn_{i,k} = sn_{i,k-1} + d_{i,k}(a_k)$.

Note that these rules are completely generic, in the sense that they are the same for any linear domain. In order to fully specify the utility functions for a specific domain, we only need to plug in the values $d_{i,j}(a)$ for every issue-value $a \in I_j$ (which we have represented in GDL by `(util ?r ?j ?v ?u)`). Therefore, we need to include a list that specifies all these values one by one, as follows:

```
(util player1 1 1  0)
(util player1 1 2 25)
(util player1 1 3 39)
...
```

These three example propositions represent the statements $d_{1,1}(a_1) = 0$, $d_{1,1}(a_2) = 25$, and $d_{1,1}(a_3) = 39$ where a_1, a_2 and a_3 are all elements of I_1. Clearly, if there are n issues, and each issue has m values, and there are 2 agents, then there are in total $2mn$ such rules required. Furthermore, we need to add two more propositions to define the reservation values:

```
(resval player1 10)
(resval player2 35)
```

All the other rules that are necessary to fully specify the structure of the graph underlying the game $G_{\mathcal{C}}$ are completely domain-independent. Therefore, we have been able to create a template GDL file which can be used for any linear domain, and for which one only needs to plug in the $2mn$ propositions of the form above and the 2 propositions stating the reservation values, in order to obtain the full description of $G_{\mathcal{C}}$. An example of such a complete GDL description is given in Appendix A.

In Genius, the linear domains are described as a table in xml format. This table simply lists the values of each $d_{i,j}(a)$, as well as the two reservation values. This roughly looks as follows:

```
<issue index="1" name="Maker">
  <item index="1" value="Apple" evaluation="0"></item>
  <item index="2" value="Samsung" evaluation="25"></item>
```

```
    <item index="3" value="Huawei" evaluation="39"></item>
  </issue>
```

Theorem 2 *Let $\mathcal{C}$ be a linear negotiation domain with n issues, and for which the largest issue has size m. Then, a description of $\mathcal{C}$ in the format of Genius can be converted to a GDL description of $G_{\mathcal{C}}$ in $O(mn)$ time and this GDL description will have $O(mn)$ size.*

Proof In order to perform the conversion, we simply need to convert every xml tag of the form `<item index="3" value = "..." evaluation="12"/>`, into a GDL rule of the form `(util player1 2 3 12)`. Clearly, the conversion of one such tag can be done in constant time. We need to do this for every value of every issue, and for both players, so there are at most $2mn$ such xml tags to convert (plus two more tags for the reservation values). Next, we need to plug them into the template GDL file. This template only needs to be created once and can be reused every time we wish to convert a linear domain, so the creation of the GDL template does not contribute to the conversion time. □

We have verified that the resulting GDL file is indeed a correct description of $G_{\mathcal{C}}$ by letting the MCNS algorithm negotiate within the Genius framework with a number of existing agents. In order to make this possible we wrapped our algorithm in a "Genius Adapter", which translated all communication between our agent and Genius back and forth between GDL and Genius format. Indeed, our algorithm was able to make correct proposals and come to profitable agreements with its opponents.

There is, however, one caveat. In Genius the utility values can take on any rational number between 0 and 1, while In GDL they can only be represented as integers between 0 and 100. Of course, we can linearly rescale the utilities to map the values from $[0, 1]$ to $[0, 100]$, but we still need to round the results off to integers, causing some loss of precision.

Another detail we should point out, is that in most negotiation domains described in the literature, including those in Genius, the opponent's utility is unknown, while in Game Theory and GGP the opponent's utility is usually assumed perfectly known. In our case, we can choose either way, because we can choose whether or not to include the rules that define the opponent's utility in the GDL description.

8.2 Hypercube Domains

To convert the Hypercube Domains of Genius to GDL format we take the same approach as for the linear domains. We use a template GDL file in which we only need to plug in a few domain-dependent propositions that define the parameters of the domain.

The template is almost identical to the one for linear domains, except that the rules of Figure 10 that define the utility functions have been replaced by the rules displayed in Figure 11. We see they look very similar. The main difference is that the summation is not over issues, but rather over constraints. The constraints themselves need to be be plugged in for every new domain. Figure 12 shows an example of two such constraints. In Genius format the same two constraints would look like the ones in Figure 13.

Theorem 3 *Let $\mathcal{C}$ be a hypercube domain with n issues, and with k constraints. Then, a description of $\mathcal{C}$ in the format of Genius can be converted to a GDL description of $G_{\mathcal{C}}$ in $O(nk)$ time and this GDL description will have $O(nk)$ size.*

```
;; Only one of the two players defected
(<= (goal ?d 100)(true (defected ?d))(not(true (defected ?c)))(role ?c))
(<= (goal ?c   0)(true (defected ?d))(not(true (defected ?c)))(role ?d))

;; Both players defected
(<= (goal ?d ?rv)(true (defected player1))(true (defected player2))(resval ?d ?rv))

;; Both players cooperated
     ;; a player with no constraints will always have goal 0.
(<= (goal ?c 0)(numConstraints ?c 0)(not(true (defected player1)))(not(true (defected player2))))

;; Here, (sum_util ?p ?n ?u) represents the fact that the sum of the utility from the first ?n constraints equals ?u
(<= (goal ?p ?g)(sum_util ?p ?n ?g)(numConstraints ?p ?n)(not(true (defected player1)))(not(true (defected player2))))
(<= (sum_util ?p 1 ?u)(util_current_value ?p 1 ?u))
(<= (sum_util ?p ?n ?sn)(succ ?m ?n)(sum_util ?p ?m ?sm)(util_current_value ?p ?n ?u)(plus ?sm ?u ?sn))

;;the utility ?u that player ?p obtains when constraint ?c is satisfied.
(<=(util_current_value ?p ?c ?u)(satisfied ?p ?c)(value ?p ?c ?u))
(<=(util_current_value ?p ?c  0)(not(satisfied ?p ?c))(constraintExists ?p ?c))

(<= (constraintExists ?p ?c)(numConstraints ?p ?n)(smallerOrEqual ?c ?n))
```

Fig. 11: GDL rules for nonlinear 'hypercube' utility functions.

```
;;  CONSTRAINT 1:
( <= (satisfied player1 1)
        (true (currentContract 2 ?x1)) (greaterOrEqual ?x1 7) (smallerOrEqual ?x1 9)
        (true (currentContract 4 ?x2)) (greaterOrEqual ?x2 2) (smallerOrEqual ?x2 7)
)
;;constraint 1 of player1 yields 50 points, if satisfied.
(value player1 1 50)

;;  CONSTRAINT 2:
( <= (satisfied player1 2)
        (true (currentContract 7 ?x3)) (greaterOrEqual ?x3 0) (smallerOrEqual ?x3 6)
)
;;constraint 2 of player1 yields 30 points, if satisfied.
(value player1 2 30)
```

Fig. 12: Two constraints for a hypercube domain defined in GDL.

```
<hyperRectangle utility="50">
   <INCLUDES index="2" min="7" max="9"/>
   <INCLUDES index="4" min="2" max="7"/>
</hyperRectangle>
<hyperRectangle utility="30">
   <INCLUDES index="7" min="0" max="6"/>
</hyperRectangle>
```

Fig. 13: Two constraints for a hypercube domain defined in Genius.

Proof In Genius, each constraint contains one xml tag for each issue that is involved in the constraint, so it can consist of at most n tags. This means that we need to convert at most nk tags. Converting a tag of the form `<INCLUDES index="2" min="7" max="9"/>`
into the following three GDL predicates

`(true (currentContract 2 ?x1))(greaterOrEqual ?x1 7)(smallerOrEqual ?x1 9)`

clearly can be done in constant time, and it has constant size. □

9 Colored Trails in GDL

The Colored Trails game was designed to model complex negotiation domains for which the contract space is not simply an unconstrained Cartesian product of finite sets. The negotiation domain of Colored Trails is a typical example of a Strategic Negotiation Domain.

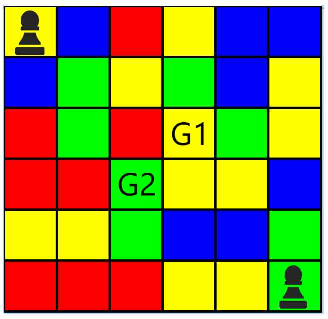

Fig. 14: Example of a 6x6 colored trails board. The two players have their pawns at the top-left and bottom-right corners respectively. Their respective goal squares are at the center of the board, indicated with $G1$ and $G2$.

It is important to understand, however, that it is not a single uniquely defined game with a unique ruleset, but rather a parametrized family of games, which leaves researchers with enough freedom to tweak the details of the game for the purpose of their own research.

9.1 General Description of Colored Trails

A Colored Trails game G_{ct} is a game that takes place on a square $m \times m$ grid, in which each square of the grid has a color assigned to it from some predefined set of colors, say $\{red, blue, yellow, green\}$. Each player is endowed with a number of chips and each of these chips also has a color, from the same set. Furthermore, each player has an *initial square* and a *goal square* on the grid (see Figure 14). Each player aims to move a pawn from their initial square to their goal square. However, each time a player moves their pawn, that player loses a chip of the same color as the square the pawn is moving to. If the player does not have any chip of that color then they cannot make that move.

The idea is that the players initially do not have the chips of the right colors to be able to reach their respective goals, but they can acquire the right chips by trading them with each other. This trading of chips can be seen as a Strategic Negotiation Domain over G_{ct}.

Although the general aim of the players is to reach their goal squares, there is no unique generally accepted utility function for Colored Trails that is used by all researchers. Instead, researchers are free to define any kind of utility functions they like. Typically, however, they would satisfy the following two conditions:

- The closer a player's pawn gets to the goal square, the higher that player's utility.
- The more chips the player owns at the end of the game, the higher that player's utility.

Also, there is no general termination criterion, but typically a game would end if neither of the two players is able or willing to move their pawn any further, and neither of the two players is willing to exchange any more chips.

This means that in order to uniquely define an instance of a Colored Trails game which can be described in GDL one needs to choose the following parameters:

- the grid size
- the set of colors
- the assignment of colors to squares

- the players' initial squares
- the players' goal squares
- the players' chip endowments
- the players' utility functions
- the termination conditions

Furthermore, in order for agents to be able to negotiate in this domain, one also needs to fix the following:

- which types of agreements can be made (i.e. a definition of the agreement space)
- when the negotiations take place (e.g. before each turn, or only before the first turn).

9.2 Colored Trails as an Extensive Form Game

We now explain how we model Colored Trails as an extensive-form game in the sense of Definition 7.

A state, in our model, consists of the grid, with colors assigned to its squares, the locations of the targets and the pawns, and a set of colored chips for each player. We define three types of actions: a *pawn move*, a *chip donation*, and a *waive*. A pawn move means that a player moves its pawn to one of the four squares adjacent to its current position. A chip donation means that a player gives a chip to the other player. A 'waive' means that the player decides to do nothing.[9]

For the utility functions we have chosen the following expression:

$$u_i(t) = \begin{cases} 5c_i + 50 - 10s_i & \text{if } s_i \leq 3 \\ 5c_i & \text{otherwise} \end{cases}$$

where c_i is the number of chips owned by player α_i in terminal state t and s_i is the number of steps the player's pawn is away from its goal (so if the pawn is at its goal square then $s_i = 0$).

In order to ensure that the game terminates after a finite number of rounds we impose the rule that the game ends after a fixed number of rounds (e.g. after 40 moves), or if both players play the 'waive' action in two consecutive rounds.

We have implemented a number of such instances of Colored Trails in GDL. An example of such a GDL description is given in Appendix B. To generate such a GDL description we follow the same recipe as with the Genius domains: we use a template that contains all the rules that are identical for all instances of Colored Trails, and then we plug in a small set of rules which define the parameters of that specific instance.

Theorem 4 *Suppose we have an instance of Colored Trails with a grid of size $m \times m$, with c colors. Then the size of the GDL description of that instance will be $O(m^2 + c)$.*

Proof As one can see in Appendix B, the domain-dependent rules that need to be plugged into the template contain one rule to indicate the color of each square of the grid, so there are m^2 such rules. Furthermore, they contain one rule to define each color, one rule to indicate how many chips player 1 has of that color, and another such rule for player 2, so there are $3c$ such rules. □

[9] This is not exactly the same as a noop-move that is commonly used in GDL to model turn-taking games. A noop-move represents the case that the player does not move because it is not their turn, while the waive move means that it is that player's turn, but the player chooses to do nothing.

We are not aware of any standard format to describe Colored Trails instances, so we will not make any formal claims about conversion to GDL. However, if we can encode an instance in GDL in $O(m^2 + c)$ size, then any other existing format should be able to do the same, so it is reasonable to believe it could be converted to GDL in $O(m^2 + c)$ time.

9.3 Negotiations over a Colored Trails Game

As explained above, the Colored Trails framework in general does not specify what exactly the players are allowed to propose to one another. In order to keep everything as similar as possible to the other domains in this paper we model the negotiations as a Branch Negotiation Domain $\mathcal{B}(G_{ct})$ where G_{ct} is an instance of Colored Trails as described in Section 9.2. This means that a player may propose to the other player a sequence of 'chip donations', 'pawn movements', and 'waives'. There is, however, nothing that prevents us from using any other type of Strategic Negotiation Game over G_{ct}. For example, one could choose to only allow the negotiation of chip-exchanges.

Another question is *when* exactly the players are allowed to negotiate. Can they only negotiate and exchange chips *before* they start moving their pawns? Or, are they allowed to restart negotiations each time after either of the two players moves its pawn (e.g. similar to the game of Diplomacy)? Again, any choice is fine and for our experiments it does not matter, so we leave this question open.

10 Experiments

In Sections 8 and 9 we have shown that it is possible to generate GDL descriptions of Genius and Colored Trails instances efficiently. The next question we want to answer is whether these GDL descriptions can also be *parsed* efficiently by a negotiation algorithm. For this, we have designed a number of experiments with which we aim to answer the following two questions. Given only the GDL description of a game G,

– *how quickly can an agent detect its reservation value for the negotiation domain $\mathcal{B}(G)$?*
– *how quickly can an agent discover and evaluate possible agreements in $\mathcal{B}(G)$?*

We repeated these experiments for three types of domains: linear Genius domains, non-linear (hypercube) Genius domains, and Colored Trails domains.

The first question is important, because extracting the reservation values of $\mathcal{B}(G)$ from a GDL description is a non-trivial task. Recall from Section 5 that they are the minimax or maximax values of G, and determining these values is, in general, an exponential problem, so we cannot simply apply a brute-force minimax algorithm. Furthermore, we should stress that we need to use an algorithm that works for *any* game so it cannot use any knowledge specific to the type of domains. For example, if an agent knows that a given GDL description represents a linear domain $\mathcal{C}$ it could use knowledge of the structure of $G_{\mathcal{C}}$ to find the minimax values, but in our experiments we do not do that, because that would be cheating, since the whole point of the declarative approach is that one cannot use any such meta-information. A typical domain-independent algorithm that meets these requirements is MCTS [35]. Furthermore, MCTS can be used just as well to calculate the pessimistic maximax values, in case the opponent utility is known.

We therefore implemented a Score-Bounded MCTS algorithm [5], with UCT [35] heuristics. The UCT constant was set to 40, as this seems to be commonly chosen value in GGP (e.g. [17]). Although many improvements on these heuristics have been proposed in the literature, such as RAVE [21], MAST, FAST and PAST [17], we did not use any of these techniques and for the random playouts we just used ordinary, uniform, sampling. After all, the purpose of this work is not to explore state-of-the-art MCTS algorithms, but rather to show that the reservation values of negotiation domains described in GDL can be determined with a relatively simple algorithm.

The second question is important, because it tells us how fast any search algorithm can explore the agreement space. The more deals an agent can evaluate within the limited time it has, the more likely it is that it finds any good ones. To answer this question we implemented a simple depth-first search (DFS) algorithm that iterates over all branches of G. Again, this is of course a very simplistic approach, but this is intentional. More sophisticated algorithms like Genetic Algorithms or Simulated Annealing would typically still need to evaluate thousands of deals, so we want to have an idea of the speed that can be achieved.

We should stress that the two algorithms we used in these experiments (MCTS and DFS) were used on *all three types of domains, without changing even a single parameter*. The implementations of both algorithms are based on a PropNet (see Section 7). Although this PropNet was implemented by ourselves, it was largely based on the one that comes with the GGP Java code base.[10]

We did not use a full negotiation algorithm (like MCNS) for our experiments, and we did not compare our algorithms with any existing negotiating agents, because that would highly depend on details such as its bidding-, acceptance-, and opponent-modeling strategies (a.k.a. BOA strategies [3]), which have nothing to do with the subject of this paper. In principle, any existing BOA strategies can be used in combination with our GDL-based exploration algorithms. Therefore, instead of using the full MCNS algorihtm, we just extracted its main components that can be used to answer our two questions. Specifically, the MCTS algorithm we use to determine reservation values is exactly the same as the one underlying the MCNS algorithm. Furthermore, our depth-first algorithm generates move sequences using exactly the same PropNet and StateMachine implementations as the MCNS algorithm.

All experiments were implemented in Java and executed on a laptop with Intel Core i7-8750H@2.20Ghz CPU and 32 GB RAM.

10.1 The Linear Domains from ANAC 2012

We have converted each of the 24 linear Genius domains that were used for ANAC 2012 into GDL, according to the recipe of Section 8.1 and we applied both our MCTS algorithm and our depth-first search to all these domains. The results are displayed in Table 2. We only show the 5 largest domains, because the results for the smaller domains were negligible.

We see that *in all cases the equilibrium value could be found in less than a millisecond, independent of the size of the domain*.

The reason that MCTS can do this so quickly, is that it only needs to evaluate a few states to come to this conclusion. As can be seen in Fig. 6 it is enough to know that the values r_1, r_2, q_1, and q_2 are all between 0 and 100 (which is a standard assumption in GGP) to determine the minimax values of the TTPD. This same principle applies to games of the form of Fig. 9, so it is sufficient for the algorithm to evaluate only the three terminal nodes at depth 2.

[10] https://github.com/ggp-org/ggp-base

Domain	**Size**	**Exploration** (ms.)	**R.V.** (ms.)
Energy	390,625	3,135 $\pm$ 8	< 1
Travel	188,160	1,324 $\pm$ 4	< 1
Supermarket	112,896	791 $\pm$ 2	< 1
EnergySmall	15,625	120 $\pm$ 1	< 1
Car	15,625	106 $\pm$ 1	< 1

Table 2: The five largest domains of ANAC 2012. **Size**: the number of possible deals in each domain. **Exploration**: The time to discover and evaluate all possible deals (averaged over 600 repetitions) plus standard error. **R.V.**: The time to determine the reservation value of α_1 (60 repetitions). All times are indicated in milliseconds.

We also see that even in the largest domain the depth-first search was able to find and evaluate all possible deals in just over 3 seconds. For comparison, the deadlines in ANAC 2012 were set at 3 *minutes*.[11]

10.2 The Hypercube Domains from ANAC 2014

We have repeated the above experiments with the hypercube domains used for ANAC 2014. However, in this case the domains were much too large for exhaustive exploration of all possible deals, because the number of deals varied between 10^{10} and 10^{50}. Therefore, instead, we measured the time required to explore 1 million deals. The results are displayed in Table 3.

We see that evaluating potential deals in these domains is a lot slower than with the linear domains (most of them between 20,000 and 50,000 deals per second, versus 100,000 deals per second in the linear case) and that the time to evaluate those deals varies highly between the instances. Overall, the required time seems to increase with the size of the contracts (in these domains each issue always had exactly 10 values, so if a domain has size 10^{30} it means that each contract involved 30 issues). Indeed, one would expect the time to evaluate a single deal (or a million deals) to be linear in the number of issues, because each deal corresponds to a branch and the algorithm needs to iteratively generate the branch state-by-state until it reaches a terminal state, and, as shown in Figure 9, the length of a branch is exactly the number of issues plus 2. Unfortunately, Genius does not have enough domains of various sizes for us to verify this expected linear behavior.

We also see that the time necessary to determine the reservation values is again very small, albeit a bit larger than in the case of the linear domains. Interestingly, we see this time that the time required does increase for larger instances. We argue that this is caused not so much because domain is larger, but rather because the representation of the states is larger. After all, just as for the linear domains, the number of states that need to be explored to find the reservation values is independent of the domain size, but the representation of the states does get larger with increasing number of issues, which means that the PropNet needs more time to determine the legal actions. In our experiment with the linear domains this was not apparent, because the number of issues in those domains was simply too small to notice this.

[11] http://anac2012.ecs.soton.ac.uk/

Domain	**Size**	**Exploration** (ms.)	**R.V.** (ms.)
10issuesDiscounted_profile-1	10^{10}	22,485 ± 544	< 1
10issuesDiscounted_profile-2	10^{10}	19,119 ± 475	< 1
10issuesDiscountedwithRV_profile-1	10^{10}	21,302 ± 703	< 1
10issuesDiscountedwithRV_profile-2	10^{10}	22,498 ± 773	< 1
10issues_profile-1	10^{10}	18,419 ± 453	2
10issues_profile-2	10^{10}	16,066 ± 286	1
10issueswithRV_profile-1	10^{10}	12,880 ± 296	< 1
10issueswithRV_profile-2	10^{10}	25,215 ± 1,308	< 1
30issuesDiscounted_profile-1	10^{30}	36,786 ± 1,485	5
30issuesDiscounted_profile-2	10^{30}	25,681 ± 1,590	4
30issuesDiscountedwithRV_profile-1	10^{30}	35,430 ± 1,505	5
30issuesDiscountedwithRV_profile-2	10^{30}	50,302 ± 2,897	4
30issues_profile-1	10^{30}	30,590 ± 1,163	4
30issues_profile-2	10^{30}	29,339 ± 1,592	4
30issueswithRV_profile-1	10^{30}	35,728 ± 1,688	4
30issueswithRV_profile-2	10^{30}	19,299 ± 878	5
50issuesDiscounted_profile-1	10^{50}	61,884 ± 2,457	7
50issuesDiscounted_profile-2	10^{50}	56,864 ± 3,083	8
50issuesDiscountedwithRV_profile-1	10^{50}	49,426 ± 2,752	8
50issuesDiscountedwithRV_profile-2	10^{50}	40,819 ± 1,259	8
50issues_profile-1	10^{50}	45,807 ± 2,540	8
50issues_profile-2	10^{50}	43,459 ± 1,821	9
50issueswithRV_profile-1	10^{50}	27,528 ± 1,670	7
50issueswithRV_profile-2	10^{50}	36,458 ± 2,188	8

Table 3: The hypercube domains from ANAC 2014. **Size**: the number of possible deals in each domain. **Exploration**: The time in milliseconds to discover and evaluate 1,000,000 deals (averaged over 20 repetitions) plus standard error. **R.V.**: The time in milliseconds to determine the reservation value of α_1 (averaged over 50 repetitions). In all cases the standard error was smaller than 1, so we do not display it.

10.3 Colored Trails

Finally, we performed the same experiments on the Colored Trails game. In this case, instead of using an existing database of Colored Trails instances we randomly generated a number instances ourselves. Each domain consisted of a 6×6, 7×7, or 8×8 grid, with 4 colors, and for each domain with grid size $m \times m$ both players had $m + 1$, or $m + 2$ chips assigned to them. The maximum number of rounds was always set to 40. The initial squares of the two players were set at the top-left and bottom-right squares respectively, while their goal squares were set at the center of the grid (as in Fig. 14). The colors of each square and each chip were chosen randomly. However, we did make sure that each instance satisfied the following two criteria:

1. If the players did not make any agreement then the theoretically highest possible score could only be obtained by not moving at all.
2. There was always at least one possible deal in the agreement space that allowed each player to reach their target.

In [23] such domains were called *all dependent* boards. This was ensured by simply generating random domains, then checking for each of them whether by coincidence it satisfied these criteria, and discarding those that did not. Unlike the experiments with the Genius domains, we assumed that the players knew each others' utility functions, since that seems to be the more common approach with Colored Trails. We performed our experiments on 30 such domains. The results are shown in Table 4.

We see that the time to evaluate 1 million deals in Colored Trails is comparable to the time it takes in a Hypercube domain. In fact, the time for these Colored Trails instances seems to fall roughly between the times for hypercube domains with 30 issues and hypercube domains with 50 issues. This makes sense, because most branches in the Colored Trails game have a length of 40 rounds, so this seems consistent with our earlier stated hypothesis that the evaluation time increases linearly with the length of each branch.

Evaluating the reservation values also takes a lot more time than with the Genius domains. This is no surprise, because for Colored Trails we did not apply the same trick of using the TTPD to set the reservation values, as we did with the Cartesian domains. This means that in the worst case the algorithm may need to explore the entire game tree in order to find the reservation values (although this clearly did not happen in the experiment).

Furthermore, we note that the MCTS algorithm always found the correct reservation values, except for the fifth instance with grid size 6×6 and 7 chips per player. For this instance it only converged to the correct values in 5 out of 10 times. In the cases it did converge to the right value it did so in around 5 seconds. In the other cases it also converged in roughly the same time, but to the wrong value. At this point it is not clear to us what makes this instance so special that the algorithm only fails on that one (and only some times). We leave this for future work.

Finally, it is worthy to note that the time necessary to find the reservation values varies wildly among the domains but it does not seem to have any correlation with the grid size or the number of chips. Again, we leave the question why this is the case open for future investigation.

11 Formal Proofs

The purpose of this section is to prove Theorem 1. In order to do this, however, we first need to state a number of formal definitions, and prove a number of intermediate lemmas.

The following lemma is not very surprising, and its proof is just a straightforward application of the well-known technique of backward induction. Furthermore, this result can also be seen easily from Figure 6. Nevertheless, we think it is important to state it here, because it plays a role later on in the proof of Theorem 1.

Lemma 1 *The minimax values of the Turn-Taking Prisoner's Dilemma are given by* $\tilde{u}_i(G) = r_i$.

Proof We first calculate $\tilde{u}_1(G)$. That is, the value that α_1 would obtain if α_2 tried to minimize u_1 (the left-hand side of Fig. 6).

At v_d the active player α_2 can choose between t_{dd} and t_{dc} as the next state. For these two states we have $u_1(t_{dc}) = 100$ and $u_1(t_{dd}) = r_1$. Since $r_1 < 100$ we have $v_d^{min,1} = t_{dd}$. This means we have:

$$\tilde{u}_1(v_d) := \tilde{u}_1(v_d^{min,1}) = u_1(t_{dd}) = r_1.$$

At v_c α_2 can choose between t_{cd} and t_{cc}. For these two states we have $u_1(t_{cd}) = 0$ and $u_1(t_{cc}) = q_1$. Since $0 < q_1$ we have $v_c^{min,1} = t_{cd}$. This means we have:

$$\tilde{u}_1(v_c) := \tilde{u}_1(v_c^{min,1}) = u_1(t_{cd}) = 0.$$

Grid Size	**Chips**	**Instance ID**	**Exploration** (ms.)	**R.V.** (ms.)
		instance 1	42,378 ± 1,031	4,037 ± 103
		instance 2	41,575 ± 835	4,989 ± 119
6 × 6	7	instance 3	40,864 ± 458	9,884 ± 218
		instance 4	43,440 ± 1,044	4,003 ± 141
		instance 5	44,596 ± 933	–
		instance 1	41,195 ± 800	5,380 ± 108
		instance 2	41,050 ± 576	4,725 ± 127
6 × 6	8	instance 3	41,193 ± 870	5,659 ± 144
		instance 4	41,533 ± 756	7,782 ± 146
		instance 5	42,264 ± 858	5,426 ± 193
		instance 1	43,155 ± 779	4,721 ± 299
		instance 2	42,797 ± 847	3,940 ± 145
7 × 7	8	instance 3	45,915 ± 1,451	6,314 ± 163
		instance 4	42,025 ± 765	3,248 ± 213
		instance 5	43,569 ± 917	10,155 ± 120
		instance 1	42,504 ± 637	7,563 ± 241
		instance 2	42,284 ± 775	5,107 ± 216
7 × 7	9	instance 3	43,135 ± 621	5,639 ± 242
		instance 4	42,124 ± 1,006	8,814 ± 236
		instance 5	41,675 ± 511	4,729 ± 170
		instance 1	44,938 ± 1,038	3,340 ± 156
		instance 2	42,841 ± 783	7,786 ± 370
8 × 8	9	instance 3	42,970 ± 604	5,484 ± 429
		instance 4	44,489 ± 427	7,388 ± 245
		instance 5	43,683 ± 532	9,705 ± 211
		instance 1	44,005 ± 819	3,296 ± 202
		instance 2	42,591 ± 462	5,486 ± 249
8 × 8	10	instance 3	43,862 ± 750	3,747 ± 243
		instance 4	43,363 ± 716	3,023 ± 154
		instance 5	45,317 ± 861	6,489 ± 237

Table 4: Thirty randomly generated instances of Colored Trails. **Grid Size**: the number rows and columns of the grid. **Chips**: The number of chips initially assigned to each player. **Instance ID**: For each grid size and chip number we created 5 different instances, which we identify with a number from 1 to 5. **Exploration**: The time to discover and evaluate all possible deals (averaged over 20 repetitions) plus standard error. **R.V.**: the time to determine the reservation value of α_1 (10 repetitions). All times are indicated in milliseconds.

Finally, at v_0 the the active player α_1 can choose between v_d and v_c. We have shown above that $\tilde{u}_1(v_d) = r_1$ and $\tilde{u}_1(v_c) = 0$, and Since $0 < r_1$ we have $v_0^{max,1} = v_d$. This means we have:

$$\tilde{u}_1(v_0) := \tilde{u}_1(v_0^{max,1}) = \tilde{u}_1(v_d) = r_1. \tag{2}$$

Next, in a similar fashion we calculate $\tilde{u}_2(G)$. That is, the value that α_2 would obtain if α_1 tried to minimize u_2 (the right-hand side of Fig. 6).

At v_d the active player α_2 can choose between t_{dd} and t_{dc}. For these we have $u_2(t_{dd}) = r_2$ and $u_2(t_{dc}) = 0$. Since $0 < r_2$ we have $v_d^{max,2} = t_{dd}$. This means we have:

$$\tilde{u}_2(v_d) := \tilde{u}_2(v_d^{max,2}) = u_2(t_{dd}) = r_2.$$

At v_c α_2 can choose between t_{cd} and t_{cc}. For these two states we have $u_2(t_{cd}) = 100$ and $u_2(t_{cc}) = q_2$. Since $q_2 < 100$ we have $v_c^{max,2} = t_{cd}$. This means we have:

$$\tilde{u}_2(v_c) := \tilde{u}_2(v_c^{max,2}) = u_2(t_{cd}) = 100.$$

Finally, at v_0 the the active player α_1 can choose between v_d and v_c. We have shown above that $\tilde{u}_2(v_d) = r_2$ and $\tilde{u}_2(v_c) = 100$, and Since $0 < r_1$ we have $v_0^{min,2} = v_d$. This means we have:

$$\tilde{u}_2(v_0) := \tilde{u}_2(v_0^{min,2}) = \tilde{u}_2(v_d) = r_2. \tag{3}$$

The lemma then follows from Equations (2) and (3) and the definition that $\tilde{u}_i(G) := \tilde{u}_i(v_0)$.
□

Lemma 2 *If B is a branch of some game G then its minimax values are exactly the utility values of its unique terminal state t, i.e. $\tilde{u}_i(B) = u_i(t)$.*

Proof Each branch B obviously has a unique terminal state t (Def. 13) and therefore, obviously, the utility values $u_i(t)$ are the only possible values the players can obtain.

11.1 Isomorphism and Weak Isomorphism

In the following we define several notions of equivalence between negotiation domains. Intuitively, when we say that two negotiation domains A and B are equivalent, we mean that for a negotiation algorithm it should not make any difference whether it receives a description of domain A or domain B as its input. In both cases it should make exactly the same proposals (in practice, however, one of the two representations might allow for a more efficient exploration of the agreement space than the other, which could still lead to differences).

Let $A = \langle \Omega^A, c^A, U_1^A, U_2^A \rangle$ and $B = \langle \Omega^B, c^B, U_1^B, U_2^B \rangle$ be two negotiation domains. If for each contract $x \in \Omega^A$ there is a corresponding contract $y \in \Omega^B$ which yields exactly the same utility values for both players, and vice versa, and the reservation values are also the same, that is:

$$\forall i \in \{1,2\}\ \forall o \in \mathcal{O}^A \quad : \quad U_i^A(o) = U_i^B(f(o))$$

then clearly we should consider A and B equivalent. In that case we say there is a *utility preserving bijection* from $\mathcal{O}^A$ to $\mathcal{O}^B$ (recall that $\mathcal{O}^A := \Omega^A \cup \{c^A\}$ and $\mathcal{O}^B := \Omega^B \cup \{c^B\}$). See Figure 15 for an example.

This notion of equivalence is, however, too strict for us, because we argue that applying a strictly increasing linear transformation to the utility functions does not essentially change a negotiation domain. After all, it should not matter whether you measure utility on a scale from 0 to 1 or on a scale from 0 to 100. In fact, this is even one of the axioms that Nash used to derive his famous Nash Bargaining Solution [45]. We therefore provide a somewhat more general definition of 'utility preserving' which allows for linear rescaling (for a short discussion on *non-linear* rescalings, see Section 12.5).

Definition 15 A **linear rescaling function** is a function $\lambda : \mathbb{R} \to \mathbb{R}$ defined by $\lambda(x) = a \cdot x + b$ for some $a, b \in \mathbb{R}$ with $a > 0$.

Definition 16 Let O be some non-empty subset of $\mathcal{O}^A$ and let f be a map from O to $\mathcal{O}^B$. We say that f is **utility preserving** if there exist two rescaling functions λ_1 and λ_2, such that:

$$\forall i \in \{1,2\}\ \forall o \in O \quad : \quad U_i^A(o) = \lambda_i(U_i^B(f(o)))$$

Note that the identity function $x \mapsto x$ is also a rescaling function, so if f satisfies $\forall i \in \{1,2\}\ \forall o \in O \ : \ U_i^A(o) = U_i^B(f(o))$ then it is also utility preserving.

Definition 17 A map f from $\mathcal{O}^A$ to $\mathcal{O}^B$ is called an **isomorphism** between A and B if it is bijective, utility preserving, and maps the conflict outcome of A to the conflict outcome of B (i.e $f(c^A) = c^B$). If there exists an isomorphism between A and B then we say that A and B are **isomorphic**.

See Figure 16 for an example of two isomorphic domains.

It turns out, however, that isomorphism is a form of equivalence that is still somewhat too strict for our purposes, because we argue that, when comparing A and B, we may just as well ignore those deals that are not individually rational, because for such deals at least one of the two negotiators would never agree to it anyway (assuming perfect rationality). To take this into account, we define the notion of a *weak isomorphism* which is only required to be utility preserving on the set of *individually rational* deals (see Def. 2). We will denote the set of individually rational contracts as Ω_{rat} and we define $\mathcal{O}_{rat} = \Omega_{rat} \cup \{c\}$.

Definition 18 Let O^A and O^B be two sets such that $\mathcal{O}^A_{rat} \subseteq O^A \subseteq \mathcal{O}^A$ and $\mathcal{O}^B_{rat} \subseteq O^B \subseteq \mathcal{O}^B$. Then a map f from O^A to O^B is called a **weak isomorphism** between A and B if it is bijective, utility preserving, and $f(c^A) = c^B$. If there exists a weak isomorphism between A and B then we say that A and B are **weakly isomorphic**.

See Figure 17 for an example of two weakly isomorphic domains. Also, for comparison, in Figure 18 we show an example of two negotiation domains that are not equivalent in any of the above senses at all.

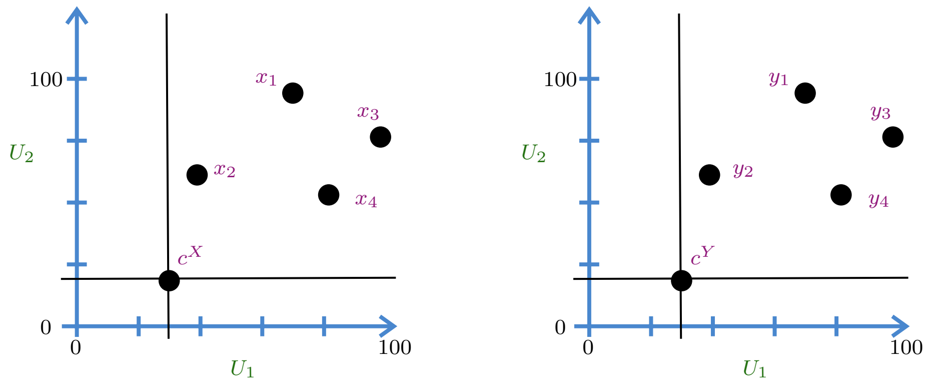

Fig. 15: Two negotiation domains X and Y which are clearly equivalent. The domain on the left has four contracts: x_1, x_2, x_3 and x_4. For each of these contracts x_i there is exactly one contract y_i in the domain on the right with exactly the same utility vector, and vice versa. Furthermore, the reservation values are also exactly the same in both domains.

11.2 Definition of $G_{\mathcal{C}}$

In order to prove Theorem 1 we will here define the game $G_{\mathcal{C}}$ (Figure 9) for which we argue that $\mathcal{B}(G_{\mathcal{C}})$ is weakly isomorphic to $G_{\mathcal{C}}$. This will then be proven in the next subsection.

Intuitively, $G_{\mathcal{C}}$ starts out like the TTPD, but with the difference that if both players play 'cooperate' the game continues for n more rounds, where n is the number of issues

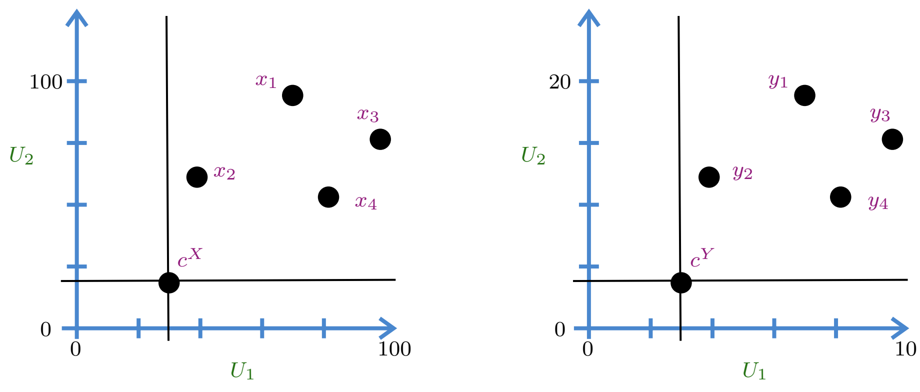

Fig. 16: Two **isomorphic** negotiation domains. For each contract x_i the corresponding contract y_i does not have the same utility vector, because the two negotiation domains measure utility on different scales. The utility vectors on the left are all in the range $[0, 100] \times [0, 100]$, while on the right they are in the range $[0, 10] \times [0, 20]$ However, the second can be transformed into the first by means of the two rescaling functions λ_1, λ_2 defined by: $\lambda_1(r) = 10 \cdot r$ and $\lambda_2(r) = 5 \cdot r$

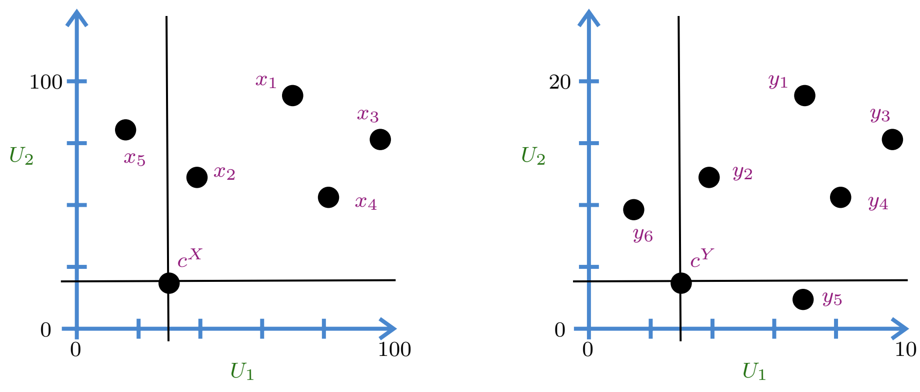

Fig. 17: Two **weakly isomorphic** negotiation domains. Although there is no bijection between the two full outcome spaces, there is a bijection, which is also utility preserving, if we ignore the irrational contracts. That is, there is a utility preserving bijection between $\mathcal{O}^X_{rat} = \{c^X, x_1, x_2, x_3, x_4\}$ and $\mathcal{O}^Y_{rat} = \{c^Y, y_1, y_2, y_3, y_4\}$.

of $\mathcal{C}$. In each of these rounds the active player chooses a value for one of the issues of the domain. Specifically, in round $j+2$ the active player picks a value a_j for issue I_j, so that the sequence of actions played between round 2 and round $n+2$ represents a complete contract $(a_1, a_2 \dots a_n) \in \Omega^{\mathcal{C}}$ of $\mathcal{C}$.

For the following, we define a **tree** as a directed acyclic graph such that there is exactly one vertex with no incoming edges (the **root**) and all other vertices have exactly one incoming edge. A **path** is a sequence of vertices $(v_0, v_1, \dots v_n)$ such that every pair of consecutive vertices is connected by an edge in the graph, (i.e. $(v_j, v_{j+1}) \in E$) and the **depth** of a vertex v is the number of edges that need to be traversed, starting at the root, to arrive at v.

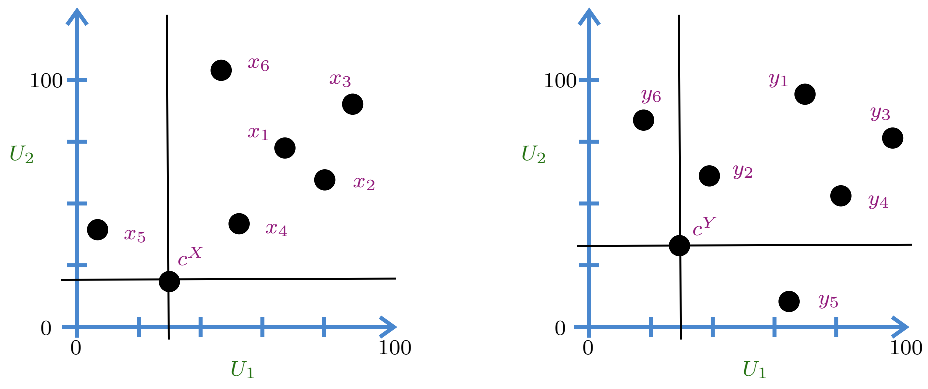

Fig. 18: Two negotiation domains that are not equivalent in any of the senses above. Although the two domains have the same size (6 contracts each) so there is a bijection between the two, this bijection is not utility-preserving because the two domains have entirely different utility vectors.

Definition 19 For any game G let M denote the map that maps every path of G to the set of moves along that path. That is:

$$M(v_0, v_1, \dots v_n) = (\ m(v_0, v_1)\ ,\ m(v_1, v_2)\ ,\ \dots\ ,\ m(v_{n-1}, v_n)\ )$$

For example, if G is the TTPD, then $M(v_0, v_c, t_{cc}) = (cooperate, cooperate)$

Definition 20 Let $\mathcal{C} = \langle \Omega^{\mathcal{C}}, c^{\mathcal{C}}, U_1^{\mathcal{C}}, U_2^{\mathcal{C}} \rangle$ be a Cartesian negotiation domain with n issues. Then we define $G'_{\mathcal{C}} = \langle V', E', v'_0, T', p', A', m', u'_1, u'_2 \rangle$ to be the unique game that satisfies the following conditions (see Table 1 and Figure 3 for an example).

- The underlying graph $\langle V, E \rangle$ is a tree.
- v'_0 is the root of the tree.
- if a vertex v of the tree has depth j with $j < n$ then:
  - v has exactly $|I_{j+1}|$ outgoing edges.
  - each outgoing edge (v, w) is labeled with a different value from issue I_{j+1}, i.e. $m(v, w) \in I_{j+1}$
  - $p(v) = j(mod\ 2) + 1$ (players alternate turns)
- if a vertex v has depth n then it has no outgoing edges (i.e. the set of terminal states T' consists of exactly the vertices with depth n).
- If $(v'_0, v_1, \dots v_n)$ denotes a path from v'_0 to some terminal state v_n, then the utility of that terminal state is given by:

$$u'_i(v_n) := U_i^{\mathcal{C}}(M(v'_0, v_1, \dots v_n)). \tag{4}$$

Note that $M(v'_0, v_1, \dots v_n)$ is a tuple $(a_1, a_2, \dots a_n)$ for which each a_j is an element of I_j. Therefore, it is an element of $I_1 \times I_2 \times \dots I_n$, which by definition is $\Omega^{\mathcal{C}}$. In other words, every path from v_0 to a terminal state v_n is associated with a contract of $\mathcal{C}$ and the utility values $u'_i(v_n)$ associated with the state v_n are defined to be exactly the utility values $U_i^{\mathcal{C}}(a_1, a_2, \dots a_n)$ of that contract.

We can now formally define the game $G_{\mathcal{C}}$ from Figure 9 as follows, by ‘gluing’ together the TTPD with $G'_{\mathcal{C}}$.

Definition 21 Let us here denote the TTPD as $G^{pd} = \langle V^{pd}, E^{pd}, v_0^{pd}, p^{pd}, A^{pd}, m^{pd}, u_1^{pd}, u_2^{pd} \rangle$ and $G'_{\mathcal{C}}$ is the game defined above (Def. 20). Then we define $G_{\mathcal{C}} = \langle V, E, v_0, p, A, m, u_1, u_2 \rangle$ as

- $V = V' \bigcup V^{pd} \setminus \{t_{cc}\}$
- $E = E' \bigcup E^{pd} \bigcup \{(v_c, v'_0)\} \setminus \{(v_c, t_{cc})\}$
- $v_0 = v_0^{pd}$
- $p(v) = \begin{cases} p'(v) & \text{if } v \in V' \\ p^{pd}(v) & \text{if } v \in V^{pd} \end{cases}$
- $A = A' \bigcup A^{pd}$
- $m(e) = \begin{cases} m'(e) & \text{if } e \in E' \\ m^{pd}(e) & \text{if } e \in E^{pd} \\ \mathit{cooperate} & \text{if } e = (v_c, v'_0) \end{cases}$
- $u_i(t) = \begin{cases} u_i^{pd}(t) & \text{if } t \in \{t_{dc}, t_{cd}\} \\ \lambda_i(u'_i(t)) & \text{if } t \in T' \\ \lambda_i(U_i^{\mathcal{C}}(c^{\mathcal{C}})) & \text{if } t = t_{dd} \end{cases}$

where the rescaling functions λ_i are chosen such that $0 < u_i(t) < 100$ for all $t \in T'$ as well as for $t = t_{dd}$.

11.3 Proof of Theorem 1

We are now finally ready to prove our main theorem. In order to prove it we will define a map f and then prove that it is a weak isomorphism between $\mathcal{B}(G_{\mathcal{C}})$ and $\mathcal{C}$.

First note that the set of outcomes $\mathcal{O}^{\mathcal{B}(G_{\mathcal{C}})}$ of $\mathcal{B}(G_{\mathcal{C}})$ can be divided into three types of outcomes:

- The conflict outcome.
- Branches B of $G_{\mathcal{C}}$ that start with both players cooperating, i.e. for which:

  $$M(B) = (\mathit{cooperate}, \mathit{cooperate}, a_1, \dots a_n)$$

  We will call these the **long branches**
- The three other branches, that end in t_{dd}, t_{dc}, and t_{cd} respectively. We will call these the **short branches**.

We then define f as follows:

$$f(o) = \begin{cases} c^{\mathcal{C}} & \text{if } o \text{ is the conflict outcome of } \mathcal{B}(G_{\mathcal{C}}) \\ (a_1, \dots a_n) & \text{if } o \text{ is a long branch, with } M(o) = (\textit{coop.}, \textit{coop.}, a_1, \dots a_n) \\ \text{undefined} & \text{if } o \text{ is a short branch} \end{cases} \tag{5}$$

We now need to prove that f is indeed a weak isomorphism. That is, we need to show the following:

- f maps the conflict outcome of $\mathcal{B}(G_{\mathcal{C}})$ to the conflict outcome $c^{\mathcal{C}}$ of $\mathcal{C}$.
- For the domain $dom(f)$ of f we have that $\mathcal{O}_{rat}^{\mathcal{B}(G_{\mathcal{C}})} \subseteq dom(f) \subseteq \mathcal{O}^{\mathcal{B}(G_{\mathcal{C}})}$.
- f is a bijection between $dom(f)$ and some set O for which $\mathcal{O}_{rat}^{\mathcal{C}} \subseteq O \subseteq \mathcal{O}^{\mathcal{C}}$.
- f is utility preserving

The first of these points does not need to be proved, because f is simply defined to map the conflict outcome of $\mathcal{B}(G_\mathcal{C})$ to the conflict outcome $c^\mathcal{C}$. We will now prove the other three points one by one.

Lemma 3 *For the domain $dom(f)$ of f we have that $\mathcal{O}^{\mathcal{B}(G_\mathcal{C})}_{rat} \subseteq dom(f) \subseteq \mathcal{O}^{\mathcal{B}(G_\mathcal{C})}$*

Proof The domain of f consists of the conflict outcome and the long branches which are indeed elements of $\mathcal{O}^{\mathcal{B}(G_\mathcal{C})}$, so we have $dom(f) \subseteq \mathcal{O}^{\mathcal{B}(G_\mathcal{C})}$. The only outcomes that are not in the domain are the short branches, which are all irrational. Therefore, all rational outcomes are in the domain of f, so we also have $\mathcal{O}^{\mathcal{B}(G_\mathcal{C})}_{rat} \subseteq dom(f)$. □

Lemma 4 *f is a bijection between $dom(f)$ and some set O for which $\mathcal{O}^\mathcal{C}_{rat} \subseteq O \subseteq \mathcal{O}^\mathcal{C}$*

Proof We can split this lemma up in two parts: 1) f is injective, 2) for its image $Im(f)$ we have $\mathcal{O}^\mathcal{C}_{rat} \subseteq Im(f) \subseteq \mathcal{O}^\mathcal{C}$. The fact that f is injective can be seen directly from its definition. The second part is true, because we can in fact show that $Im(f) = \mathcal{O}^\mathcal{C}$. To prove this, we have to show that for every contract $(a_1, a_2, \dots a_n)$ of $\mathcal{C}$ there is a long branch $(cooperate, cooperate, a_1, a_2, \dots a_n)$ in the game $G_\mathcal{C}$. This proof goes by induction.

Let $(a_1, a_2, \dots a_n) \in I_1 \times I_2, \dots \times I_n$. First, note that there must exist at least one path $(v_0, v_1, \dots)$ in $\langle V, E\rangle$ for which $m(v_0, v_1) = a_1 \in I_1$. This is because by definition of $G'_\mathcal{C}$ the initial state v_0 has $|I_1|$ outgoing edges, and each outgoing edge must have a different label chosen from I_1. So v_0 must have exactly one outgoing edge labeled with a_1. Next, suppose that for some integer k it is proven that there exists a path $(v_0, v_1, \dots v_k \dots)$ such that $M(v_0, v_1, \dots v_k \dots)$ starts with $(a_1, a_2, \dots a_k)$. We can then repeat the same argument to argue that there must be a path $(v_0, v_1, \dots v_k, v_{k+1} \dots)$ such that $M(v_0, v_1, \dots v_k, v_{k+1} \dots)$ starts with $(a_1, \dots a_k, a_{k+1})$.

So, any contract of $\mathcal{C}$ is indeed in the image of f. Furthermore the conflict outcome $c^\mathcal{C}$ is clearly also in the image of f, so all outcomes of $\mathcal{C}$ are in the image. In other words, we have $Im(f) = \mathcal{O}^\mathcal{C}$. □

Lemma 5 *f is utility preserving.*

Proof We need to prove that for all outcomes o in the domain of f we have: $U_i^{\mathcal{B}(G_\mathcal{C})}(o) = \lambda_i(U_i^\mathcal{C}(f(o)))$. Recall that the domain of f consists of the long branches and the conflict outcome, so we first prove this equality for the long branches, and then for the conflict outcome.

First, suppose that o is a long branch, with $o = (v_0, v_c, v'_0, v_1, \dots v_n)$, and $M(o) = (coordinate, coordinate, a_1, a_2, \dots a_n)$. Then we have:

$$U_i^{\mathcal{B}(G_\mathcal{C})}(o) = \tilde{u}_i(o) = u_i(v_n) = \lambda_i(u'_i(v_n)) = \lambda_i(U_i^\mathcal{C}(M(v'_0, v_1, \dots v_n))) = \lambda_i(U_i^\mathcal{C}(f(o)))$$

where the first equality is simply by definition of a Strategic Negotiation Game (Def. 12), the second equality is by Lemma 2, the third equality is by the definition of $G_\mathcal{C}$ (Def. 21), and the fourth equality is by Eq. (4). Then, noting that $M(v'_0, v_1, \dots v_n)$ is equal to $(a_1, a_2, \dots a_n)$, which in turn is exactly $f(o)$, we obtain the last equality.

Second, suppose that o is the conflict outcome of $\mathcal{B}(G_\mathcal{C})$, that is: $o = c^{\mathcal{B}(G_\mathcal{C})} = G_\mathcal{C}$. In this case we need to prove:

$$U_i^{\mathcal{B}(G_\mathcal{C})}(o) = \tilde{u}_i(G_\mathcal{C}) = u_i(t_{dd}) = \lambda_i(U_i^\mathcal{C}(c^\mathcal{C})) = \lambda_i(U_i^\mathcal{C}(f(o))).$$

The first equality is by the definition of a Strategic Negotiation Game (Def. 12). For the second equality, first note that λ_i was chosen such that for any terminal state t that can be reached from v'_0 (i.e. each $t \in T'$) we have $0 < u_i(t) < 100$. This, in turn, implies that we have $0 < \tilde{u}_i(v'_0) < 100$. Furthermore, we also have $0 < u_i(t_{dd}) < 100$. Knowing this, the proof of this equality is analogous to the proof of Lemma 1. The third equality is by definition of $G_{\mathcal{C}}$ (Def. 21), and the last equality is simply by plugging in $f(o) = c^{\mathcal{C}}$ from the definition of f. □

These three lemmas together prove that f is indeed a weak isomorphism between $\mathcal{B}(G_{\mathcal{C}})$ and $\mathcal{C}$, and therefore our main theorem is proven.

12 Discussion and Future Work

In this paper we have shown that every Cartesian Negotiation Domain $\mathcal{C}$ there is a game $G_{\mathcal{C}}$ such that the Strategic Negotiation Domain $\mathcal{B}(G_{\mathcal{C}})$ is equivalent to $\mathcal{C}$ and that in the case of the Genius domains, the game $G_{\mathcal{C}}$ can be described efficiently in GDL. Similarly, we have shown that the Colored Trails game G_{ct} can be described efficiently in GDL. This means that any negotiation algorithm that is written for strategic domains in general, can also be applied to the existing domains from Genius as well as the Colored Trails game.

The fact that $\mathcal{C}$ is only a *weakly* isomorphic to $\mathcal{B}(G_{\mathcal{C}})$ means that there will be some contracts in the strategic domain that do not exist in the Cartesian domain, but since they are irrational anyway their presence should not matter.

In the following subsections we discuss a number of issues that we left unanswered in the text, as well some issues that we leave for future work.

12.1 Proposing General Restrictions

In this work we have restricted our attention to Branch Negotiation Domains, which form a subclass of the Strategic Negotiation Domains. This is because we wanted to simulate the Genius domains in GDL and in Genius it is assumed that agents make agreements that completely fix every aspect of the contract and that yield a well-defined unique utility value for each player, which corresponds exactly to the notion of a branch.

However, we think it could be much more interesting to study Strategic Negotiation Domains in which the agents can propose any kind of restriction. This essentially means they are allowed to make partial agreements that only specify, for example, the first few moves, or that specify that only one particular move can never be made, while the agents still keep the freedom to choose between several other moves.

In order to make this generalization we need to have a language that allows the agents to specify such general contracts. One language that could be used for this purpose is the recently introduced language SGL [70]. Furthermore, we would need some way to communicate to the agents which types of restrictions they are allowed to propose (e.g. any restriction, or only branches, or some other subset of $Rest(G)$). We leave it as future work to explore how to do this.

12.2 Negotiations in Multiple Rounds

Apart from the question *what* the agents can negotiate, another important question is *when* the agents can negotiate. Is there a negotiation stage before every round (like in the game of Diplomacy), or only before the first round? If the aim is to correctly simulate the Genius domains then negotiations should only take place before the first round. But for other domains, like Colored Trails, it is more interesting to allow agents to continue negotiating before every new round.

The MCNS algorithm was designed specifically for negotiations before every new round, and it makes less sense to apply it if negotiations only take place before the first round. Therefore, if we want to implement a more general algorithm that is able to negotiate under either scenario we need a way of indicating to the agents when they are allowed to negotiate. This can actually be done quite easily, by extending GDL with just one keyword, say `negotiate`. We could then write rules of the form

```
    (<= (negotiate) (true P))
```

which would mean that for any state in which the proposition P is true, the players are allowed to negotiate before making their moves.

12.3 Arithmetic GDL

One of the main disadvantages of GDL is that it does not have direct support for arithmetics. In order to calculate utility values we needed to perform summations, and such summations have to be hard-coded in the GDL description. For example the statement $0 + 1 = 1$ is explicitly included in the listing in Appendix A, as the proposition `(plus 0 1 1)`. In fact, we had to include every possible statement of the form $a + b = c$ for all values of a, b and c between 0 and 100, yielding a list of 5151 such propositions. Although this solution is clearly inelegant, in practice it does not yield any big problems. It just makes the parsing of the domain and the calculation of the utility values slightly inefficient.

A bigger problem, is that in this way we are only able to handle integers, and that these integers have to be within reasonably small bounds (e.g. between 0 and 100). This means that the conversion from Genius to GDL is not always exact, due to rounding errors.

In order to overcome this problem, we propose to extend GDL with arithmetical operators. For example, we could define `plus` to be a keyword, so that our algorithm immediately understands that the statement `(plus 0 1 1)` is always true, without this having to be stated explicitly in the GDL description. We could then also do the same with multiplication and division, and we may even be able to use fractional numbers. This means that we would have to adapt our algorithms to understand such an extension of GDL (if it is even feasible at all to define such an extension). Another major challenge would be to define a number of restrictions of the use of these new keywords, to ensure that the number of true propositions remains finite and small enough that algorithms can still process the GDL efficiently. We leave this as future work.

12.4 Domains with Imperfect Information

Another interesting question in the field of Automated Negotiations, which has not been studied much, is how to implement negotiating agents in domains with incomplete or imperfect information about the environment. In order to define such domains we could GDL-II [62], the extension of GDL for games with incomplete information. This is another issue we leave for future work.

12.5 Non-linear Rescaling of Utility Functions

In Section 11 we argued that if we rescale a utility function by a linear transformation, the negotiation domain remains essentially the same. A legitimate question now, is whether we could also allow more general transformations.

Of course, one could argue that non-linear, but monotonous, transformation should also be allowed, because they preserve the preference ordering between the potential agreements. However, it is highly debatable whether the transformed domain in that case could still be considered 'equivalent' to the original one. For example, the new domain would in general have a different Nash Bargaining solution than the original one, and also other notions such as social welfare would change. Especially, we do not think that most existing algorithms would behave in exactly the same way on two domains if those domains differed by a non-linear transformation.

Finally, if we allowed a larger class of transformations, it would just make our notion of equivalence weaker, while our aim is to have the strongest notion of equivalence as possible. Therefore, the conclusion is that we have no reason to allow non-linear transformations.

12.6 Negotiation Domains with Hard Constraints

We have argued that one of the main reasons to use GDL is that it allows us to define negotiation domains with hard constraints between the issues. However, apart from Colored Trails, and the toy-world games in [32] we have not studied any such domains. Therefore, we aim for the future to implement several such domains. For example, we could take the linear domains from Genius and adapt them by adding a number of hard constraints.

12.7 Hypercubes with Hard Constraints

We have argued that Hypercube domains can only be used to define soft constraints between the issues, but not hard constraints. However, we should note that hypercubes can be used to describe hard constraints if we make just one small adaptation. That is, we could allow the utility functions to also take the value $-\infty$.

For example, in the Mobile Phone domain displayed in Figure 4 we could assume that the 'color' issue has 4 values: $\{Black, White, Silver, Blue\}$, and whenever a contract involves a brand and a color that are incompatible, such as $(\text{Samsung}, \text{Blue}, 16GB)$, we set its utility to $-\infty$.

This approach, however, has several disadvantages in comparison to the use of GDL. For example, suppose that each phone is available in two colors, but the two available colors are different for each brand, so in total there are six different colors. This would mean the entire domain has $3 \times 6 \times 2 = 36$ contracts, but only $3 \times 2 \times 2 = 12$ of them are valid, which is only 33% of the entire domain. This could make it very difficult for any generic algorithm to explore such a domain, because it would mainly find invalid contracts. We definitely think this problem occurs in the real world, and it is not hard to see that in some domains these numbers can be much more extreme, especially if this problem also occurs with the other issues of the domain. After all, the fraction of valid contracts decreases exponentially with the number of such issues.

Furthermore, some of the other advantages of GDL that we mentioned before would not apply to this 'Hypercube with negative infinity' approach.

12.8 Exploring Large Strategic Domains

Arguably the most important question that has been left open, is how to find good proposals in very large strategic domains. In our experiments with the hypercube domains we showed that we were able to explore over a million possible deals in less than a minute, but 1 million is still only a tiny fraction of the entire agreement space, if the total size is 10^{50}.

The agents that participated in ANAC 2014 used intelligent search algorithms such as Genetic Algorithms [29] or Simulated Annealing [47] to overcome this problem, but such algorithms are specifically targeted to Cartesian domains. For example, if we use a straightforward genetic algorithm that samples branches of the game and we apply cross-over to combine two such branches into a new one, then the resulting new branch might not represent a legal sequence of moves of the game.[12]

This same question also plays a very important role in the game of Diplomacy. Although many researchers have published work on this game [15,30,41], still, to date no one has been able to implement an agent that has convincingly solved this problem and is able to strongly outplay any non-negotiating agent [27].

We therefore argue that *the question how to find good proposals in a large strategic domain is one of the main open questions in the field of Automated Negotiations*, and we think that the use of GDL to define such domains will help answering this question, as it allows researchers to incrementally build more and more complex domains and it will allow them to test their algorithms across many different instances of such domains.

12.9 Multilateral Domains

To keep the proofs and notation simple we have restricted ourselves to bilateral negotiation domains. However, almost everything in this work was independent of the number of agents, so all of this should equally work for multilateral domains. The only real difference is that we would need to replace the Turn-Taking Prisoners Dilemma with a variant involving more than two agents.

12.10 Other Domain Description Languages

Of course, GDL is only one example of a language that is capable of describing a large class of problem instances and we are by no means claiming that GDL is the best candidate to describe negotiation domains. Therefore, in the future we would also like to experiment with other languages and compare there usefulness with GDL. Other languages that we could consider are, for example, ASP, STRIPS, and PDDL.

12.11 Category Theory

Finally, an interesting question is whether our theorems and their proofs can be formulated in the language of Category Theory. We note, for example, that the operator $\mathcal{B}$ is in fact a functor from a certain category of Games to a category of Negotiation Domains. The isomorphisms of this category of Negotiation Domains could either be the isomorphisms

[12] Of course, there may exist more sophisticated Genetic Algorithms that are able to avoid this problem, but to the best of our knowledge no one has ever tried to apply those to automated negotiations.

as defined by Def. 17, or the weak isormorphisms as defined by by Def. 18. Also, we have defined a functor in the opposite direction, which takes any Cartesian negotiation domain $\mathcal{C}$ to the game $G_{\mathcal{C}}$. Although such an investigation may not directly have much practical use for the implementation of negotiation algorithms, we still think it could be very interesting for purely theoretical reasons.

13 Acknowledgments

This work was funded by the E.U. Horizon 2020 Research and Innovation Programme, Grant Agreement No. 769142, project LOGISTAR.

Appendix A: GDL Code for Linear Domains

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;DEFINE THE ROLES
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(role player1)
(role player2)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;DEFINE THEIR POSSIBLE ACTIONS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Remain silent
(<= (input ?r noop)(role ?r))

;; The two actions for the initial prisoner's dilemma.
(<= (input ?r cooperate)(role ?r))
(<= (input ?r defect)(role ?r))

;;Set the value of a contract property
;;We can only assign the value V to property P if V is indeed a possible value for P.
(<= (input ?r  (setValue ?p ?v )) (role ?r) (valueExists ?p ?v))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; DEFINE THE BASE PROPOSITIONS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(<= (base (step ?x)) (number ?x) )
(<= (base (control ?r)) (role ?r) )

(<= (base (defected ?r)) (role ?r) )

(<= (base (currentContract ?p ?v)) (valueExists ?p ?v))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; SET THE INITIAL STATE.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(init (step 0))
(init (control player1))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; DEFINE WHO CAN DO WHAT AND WHEN.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; The non-active player has the right to remain silent.
( <= (legal ?r noop) (role ?r)(not (true (control ?r))))
```

```
;; In the first two rounds they play a prisonner's dilemma.
( <= (legal ?r cooperate)(role ?r)(true (control ?r))(true (step 0)))
( <= (legal ?r defect)(role ?r)(true (control ?r))(true (step 0)))
( <= (legal ?r cooperate)(role ?r)(true (control ?r))(true (step 1)))
( <= (legal ?r defect)(role ?r)(true (control ?r))(true (step 1)))

;; In the p-th turn the active player may set the value of the p-th issue.
( <= (legal ?r  (setValue ?p ?v) ) (true (control ?r))(true (step ?m))
        (succ ?p ?m) (valueExists ?p ?v))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; DEFINE THE EVOLUTION OF THE STATE.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;Let the step counter increase every round.
(<= (next (step ?y)) (true (step ?x)) (succ ?x ?y) )

;;Once a player has defected this should be recorded, and this fact remains true.
;; This is so that after 2 rounds we can terminate the game whenever one of the
;; two players has defected.
(<= (next (defected ?r))(does ?r defect))
(<= (next (defected ?r))(true (defected ?r)))

;;It's a turn-taking game.
(<= (next (control ?r1)) (true (control ?r2))(distinct ?r1 ?r2)(role ?r1))

;; The value of a property remains the same, unless some agent currently sets
;; some value for that property.
( <= (next (currentContract ?p ?v)) (true (currentContract ?p ?v)) (not (valueChanged ?p)))
( <= (valueChanged ?p) (does ?r (setValue ?p ?v)) )
( <= (next (currentContract ?p ?v)) (does ?r (setValue ?p ?v)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;DEFINE THE TERMINAL CONDITIONS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 'step' starts counting at 0, so it represents the number of moves that have been made.
;; if there are 10 issues, then there are 12 rounds, so when the step counter is 12
;; the game has terminated.
(<= terminal (true (step ?n))(numIssues ?m)(plus ?m 2 ?n))
(<= terminal (true (step 2))(true (defected ?r)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; GOAL RELATIONS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Only 1 player defected
(<= (goal ?r 100)(true (defected ?r))(not(true (defected ?q)))(role ?q))
(<= (goal ?q   0)(true (defected ?r))(not(true (defected ?q)))(role ?q))

;; Both players defected
(<= (goal ?r ?rv)(true (defected player1))(true (defected player2))(resval ?r ?rv))

;; Here, (sum_util ?r ?n ?u) represents the fact that the sum of the utility
;; from the first ?n issues equals ?u

(<= (goal ?r ?g)(sum_util ?r ?n ?g)(numIssues ?n)
       (not(true (defected player1)))(not(true (defected player2))))
(<= (sum_util ?r 1 ?u)(util_current_value ?r 1 ?u))
(<= (sum_util ?r ?n ?sn)(succ ?m ?n)(sum_util ?r ?m ?sm)
       (util_current_value ?r ?n ?u)(plus ?sm ?u ?sn))

;;the utility ?u that player ?r obtains when the issue with index ?p
```

```
(<=(util_current_value ?r ?p ?u)(true (currentContract ?p ?v))(util ?r ?p ?v ?u))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; PARAMETERS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; (valueExists 2 3) means that there exists an issue with index 2 and
;; it has the value 3 in its domain.
;; to check that this is true we simply need to check whether some
;; utility value is assigned to it.
(<= (valueExists ?p ?v)(util ?r ?p ?v ?u))

;; (numIssues 5) means that there are 5 issues. To check this, we need to check
;; that some issue-value pair exists with issue-index 5,
;; and that there is no such issue-value pair with issue-index 6.
(<= (numIssues ?m)(valueExists ?m ?v)(not (existsHigher ?m)))
(<= (existsHigher ?m)(valueExists ?n ?v)(succ ?m ?n))


;;START OF DOMAIN DEPENDENT RULES

;; Set the reservation values.
(resval player1 10)
(resval player2 35)

;; For each player, assign a utility value to each issue-value pair
;; Here, there are 2 players, and 3 issues. The first issue has 3 values and the other two issues have 2 value

;; PLAYER 1:
(util player1 1 1 0) ;; player 1, issue 1, value 1 --> utility value 0
(util player1 1 2 25) ;; player 1, issue 1, value 2 --> utility value 25
(util player1 1 3 39)

(util player1 2 1 20)
(util player1 2 2 30)

(util player1 3 1 18)
(util player1 3 2 30)


;; PLAYER 2:
(util player2 1 1 40) ;; player 2, issue 1, value 1 --> utility value 40
(util player2 1 2 59) ;; player 2, issue 1, value 2 --> utility value 59
(util player2 1 3 0)

(util player2 2 1 30)
(util player2 2 2 30)

(util player2 3 1 10)
(util player2 3 2 0)


;;END OF DOMAIN DEPENDENT RULES

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;   ARITHMETICS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(number 0)
(<= (number ?y) (plus 1 ?x ?y)  )

(<= (succ ?m ?n)(plus 1 ?m ?n))
```

```
;; define addition (simply explicitly list all possible additions of
;; non-negative integers that lead to a sum smaller than or equal to 101.)
(plus 0 0 0)  ;;0 + 0 = 0
(plus 0 1 1)  ;;0 + 1 = 1
(plus 0 2 2)  ;;0 + 2 = 2
(plus 0 3 3)  ;;0 + 3 = 3
...
...  (the list is too long to fully include in this paper.)
...
(plus 99 1 100) ;; 99 + 1 = 100
(plus 99 2 101)
(plus 100 0 100)
(plus 100 1 101)
(plus 101 0 101)
```

Appendix B: GDL Code for Colored Trails

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;DEFINE THE ROLES
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(role player1)
(role player2)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;DEFINE THEIR POSSIBLE ACTIONS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; give a chip of color c to the opponent.
(<= (input ?r (give ?c))   (role ?r) (color ?c))

;; move to the given square.
(<= (input ?r (moveTo ?x ?y))(role ?r)(row_index ?x)(column_index ?y) )

;; the active player chooses to do nothing.
(<= (input ?r waive) (role ?r)  )

;; the unique action for the non-active player.
(<= (input ?r noop) (role ?r)  )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; BASE Relations
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; current positions of the players:
(<= (base (position ?r ?x ?y))(role ?r)(cell ?x ?y))


;; number of chips owned of a specific color.
(<= (base (owns ?r ?c ?n))(role ?r)(color ?c)(number ?n))

;; total number of chips owned.
(<= (base (ownsTotal ?r ?n))(role ?r)(number ?n))

;; 'control' defines the active player
(<= (base (control ?r))(role ?r))

;; 'step' counts how many rounds have been played.
(<= (base (step ?n))(number ?n))
```

```
;; WaiveCounter counts how many consecutive times the 'waive' action has been played.
;; It is increased when the active player plays 'waive'
;; It is reset to 0 every time the active player makes a move other than waive.
;; The game ends when the counter is 2 (i.e. both players have played waive).
(<= (base (waiveCounter ?n))(number ?n))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; INIT Relations
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; positions of the players:
(<=(init (position ?r ?row ?column))(init_pos ?r ?row ?column))

;;ownership of chips:
(<= (init (owns ?r ?c ?n))(init_owns ?r ?c ?n))
(<= (init (ownsTotal ?r ?n))(init_ownsTotal ?r ?n))
;; The init relations are defined in terms of the functions init_owns
;; and init_ownsTotal, which are set below, in the PARAMETERS section.
;; This is to make sure that all parameters of the game are set at the same place.

;;status of the turn-taking game
(init (control player1))
(init (step 0))
(init (waiveCounter 0))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; LEGAL Relations
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; it is legal for player r to move from (x1,y1) to (x2,y2) if:
;; - r is the active player in the current round.
;; - (x1,y1) is the current position of r
;; - (x2,y2) can be reached in one horizontal or vertical step (the move is 'valid')
;; - r owns at least one chip of the same color as the target cell.
(<= (legal ?r (moveTo ?x2 ?y2)) (true (control ?r))(true (position ?r ?x1 ?y1))
(validMove ?x1 ?y1 ?x2 ?y2) (ownsColor ?r ?c) (cellHasColor ?x2 ?y2 ?c))

;; it is legal for player r to give a chip of color c to the other player if:
;; - player r owns at least one chip of that color
;; - r is the active player in the current round.
(<= (legal ?r (give ?c))(ownsColor ?r ?c)(true (control ?r)))

;; a player r 'owns' a color c if the number of chips it owns of that color is not 0.
(<= (ownsColor ?r ?c)(true (owns ?r ?c ?n))(distinct ?n 0))

(<= (validMove ?x1 ?y1 ?x1 ?y2) (succ ?y1 ?y2) (cell ?x1 ?x2)(cell ?y1 ?y2)) ;;move up
(<= (validMove ?x1 ?y1 ?x1 ?y2) (succ ?y2 ?y1) (cell ?x1 ?x2)(cell ?y1 ?y2)) ;;move down
(<= (validMove ?x1 ?y1 ?x2 ?y1) (succ ?x1 ?x2) (cell ?x1 ?x2)(cell ?y1 ?y2)) ;;move right
(<= (validMove ?x1 ?y1 ?x2 ?y1) (succ ?x2 ?x1) (cell ?x1 ?x2)(cell ?y1 ?y2)) ;;move left

;; a player r can make the 'waive' action if r is the active player.
(<= (legal ?r waive)(role ?r)(true (control ?r)))

;; a player r can make the 'noop' action if r is not the active player.
(<= (legal ?r noop)(role ?r)(not (true (control ?r))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; NEXT Relations
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; moving causes the player to change position
```

```
(<= (next (position ?r ?x ?y))(does ?r (moveTo ?x ?y)) )


;;keep position when not moving
(<= (next (position ?r ?x1 ?y1))(true (position ?r ?x1 ?y1))(not (isMoving ?r)))
(<= (isMoving ?r) (does ?r (moveTo ?x ?y)))

;;adjust chip count for specific color:
(<= (next (owns ?r ?c ?n)) (true (owns ?r ?c ?m)) (increaseChips ?r ?c)(succ ?m ?n))
(<= (next (owns ?r ?c ?m)) (true (owns ?r ?c ?n)) (decreaseChips ?r ?c)(succ ?m ?n))
(<= (next (owns ?r ?c ?n)) (true (owns ?r ?c ?n)) (not (increaseChips ?r ?c))
        (not (decreaseChips ?r ?c)))
;; if the number of chips of a given color is neither increasing nor decreasing
;;then keep the same amount.


;; moving causes the player to lose a chip:
(<= (decreaseChips ?r ?c) (does ?r (moveTo ?x ?y)) (cellHasColor ?x ?y ?c))

 ;; giving away a chip:
(<= (decreaseChips ?r ?c)(does ?r (give ?c)))

 ;; receiving a chip:
(<= (increaseChips ?r2 ?c)(does ?r1 (give ?c))(distinct ?r1 ?r2)(role ?r2))

;; adjust total chip count:
(<= (next (ownsTotal ?r ?n)) (true (ownsTotal ?r ?m)) (increaseChips ?r ?c)(succ ?m ?n))
(<= (next (ownsTotal ?r ?m)) (true (ownsTotal ?r ?n)) (decreaseChips ?r ?c)(succ ?m ?n))
(<= (next (ownsTotal ?r ?n)) (true (ownsTotal ?r ?n)) (not (increaseChipsTotal ?r))
        (not (decreaseChipsTotal ?r)))
 ;; if the number of total chips is neither increasing nor decreasing, then keep the same amount.

;; increase/decrease the total chip count if there is any color for which the
;; chip count is increased/decreased.
(<= (increaseChipsTotal ?r) (increaseChips ?r ?c))
(<= (decreaseChipsTotal ?r) (decreaseChips ?r ?c))

;;turn-taking:
(<= (next (control ?r2))(true (control ?r1))(distinct ?r1 ?r2)(role ?r2))
(<= (next (step ?n))(true (step ?m))(succ ?m ?n))

;;reset the 'waiveCounter' when the active player does not play 'waive'.
(<= (next (waiveCounter 0))(not (does ?r waive))(true(control ?r)))

;;increase the 'waiveCounter' when the active player plays 'waive'.
(<= (next (waiveCounter 1))(true (waiveCounter 0))(does ?r waive)(true (control ?r)))
(<= (next (waiveCounter 2))(true (waiveCounter 1))(does ?r waive)(true (control ?r)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; TERMINAL Relations
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; The game ends when the maximum number of rounds have been played..
(<= terminal (true (step ?ms))(max_step ?ms))

;; or when both players have played 'waive' in two consecutive rounds.
(<= terminal (true (waiveCounter 2)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; GOAL Relations
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; The utility ('goal') of a player is the number of points multiplied by 5.
(<= (goal ?r ?g) (points ?r ?p) (timesFive ?p ?g))

;; players get points from two sources: the number of chips owned and
;; the distance to the target.
(<= (points ?r ?p)(distance_points ?r ?dp)(true (ownsTotal ?r ?cp))(sum ?dp ?cp ?p))

;; 'distance points' decreases with the distance to the target.
;; Reaching the target yields 10 points.
;; (since the utility of a player is 5 times its points, reaching the goal yields
;; a utility value of 50).
(<= (distance_points ?r 10) (distance_to_target ?r 0))
(<= (distance_points ?r 8) (distance_to_target ?r 1))
(<= (distance_points ?r 6) (distance_to_target ?r 2))
(<= (distance_points ?r 4) (distance_to_target ?r 3))
(<= (distance_points ?r 0) (not (close_to_target ?r))(role ?r))

(<=  (close_to_target ?r)(distance_to_target ?r ?d)(is_small ?d))
(is_small 0)
(is_small 1)
(is_small 2)
(is_small 3)

;; calculates the player's distance to the target, given the player's current
;; position the position of the player's target.
(<=(distance_to_target ?r ?d)(true (position ?r ?x1 ?y1))
        (target ?r ?x2 ?y2)(distance ?x1 ?y1 ?x2 ?y2 ?d))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; PARAMETERS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;START OF DOMAIN-DEPENDENT RULES

;; the game ends after 40 rounds
(max_step 40)

;; the grid has 5 rows
(row_index 1)
(row_index 2)
(row_index 3)
(row_index 4)
(row_index 5)

;; the grid has 5 columns
(column_index 1)
(column_index 2)
(column_index 3)
(column_index 4)
(column_index 5)

;; we define 4 colors.
(color blue)
(color green)
(color red)
(color yellow)

;; the players start at the top-left and bottom-right corner, respectively.
```

```
(init_pos player1 1 1)
(init_pos player2 5 5)

;; both players have their target at the center of the grid.
(target player1 3 3)
(target player2 3 3)

;; the initial chip endowment of player 1.
(init_owns player1 blue 2)
(init_owns player1 green 4)
(init_owns player1 red 0)
(init_owns player1 yellow 1)
(init_ownsTotal player1 7)  ;;the sum of the previous values

;; the initial chip endowment of player 2.
(init_owns player2 blue 0)
(init_owns player2 green 2)
(init_owns player2 red 3)
(init_owns player2 yellow 2)
(init_ownsTotal player2 7)   ;;the sum of the previous values


;; The colors of the squares of the grid.
(cellHasColor 1 1 blue)
(cellHasColor 1 2 red)
(cellHasColor 1 3 red)
(cellHasColor 1 4 blue)
(cellHasColor 1 5 yellow)

(cellHasColor 2 1 blue)
(cellHasColor 2 2 green)
(cellHasColor 2 3 red)
(cellHasColor 2 4 blue)
(cellHasColor 2 5 yellow)

(cellHasColor 3 1 green)
(cellHasColor 3 2 red)
(cellHasColor 3 3 yellow)
(cellHasColor 3 4 blue)
(cellHasColor 3 5 green)

(cellHasColor 4 1 blue)
(cellHasColor 4 2 blue)
(cellHasColor 4 3 blue)
(cellHasColor 4 4 red)
(cellHasColor 4 5 red)

(cellHasColor 5 1 yellow)
(cellHasColor 5 2 yellow)
(cellHasColor 5 3 red)
(cellHasColor 5 4 blue)
(cellHasColor 5 5 green)

;;END OF DOMAIN-DEPENDENT RULES

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; FACTS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(<= (cell ?row ?column)(row_index ?row) (column_index ?column))
```

```
(number 0)
(<= (number ?n)(succ ?m ?n))

;; Calculates the manhattan distance between two squares on the grid.
;; we add the restriction that d must be small (i.e. between 0 and 3)
;; to prevent too many instantiations of this rule.
;; if d is greater than 3 we don't need this rule anyway.
(<= (distance ?x1 ?y1 ?x2 ?y2 ?d)(diff ?x1 ?x2 ?dx)(diff ?y1 ?y2 ?dy)
        (sum ?dx ?dy ?d)(row_index ?x1)(column_index ?y1)(row_index ?x2)
        (column_index ?y2)(is_small ?d))


;;sum:
(<=(sum ?x ?y ?s)(sum_assym ?x ?y ?s))
(<=(sum ?x ?y ?s)(sum_assym ?y ?x ?s))
(<=(sum_assym ?x ?y ?s)(diff_assym ?x ?s ?y))

;;the absolute difference:
(<= (diff ?x ?y ?d)(diff_assym ?x ?y ?d))
(<= (diff ?x ?y ?d)(diff_assym ?y ?x ?d))

(<=(diff_assym ?x ?x 0)(number ?x))
(<=(diff_assym ?x0 ?x1 1)(succ ?x0 ?x1))
(<=(diff_assym ?x0 ?xd ?d)(succ ?xc ?xd)(succ ?c ?d)(diff_assym ?x0 ?xc ?c))

(succ 0 1)
(succ 1 2)
(succ 2 3)
(succ 3 4)
(succ 4 5)
(succ 5 6)
(succ 6 7)
(succ 7 8)
(succ 8 9)
(succ 9 10)
(succ 10 11)
(succ 11 12)
(succ 12 13)
(succ 13 14)
(succ 14 15)
(succ 15 16)
(succ 16 17)
(succ 17 18)
(succ 18 19)
(succ 19 20)
(succ 20 21)
(succ 21 22)
(succ 22 23)
(succ 23 24)
(succ 24 25)
(succ 25 26)
(succ 26 27)
(succ 27 28)
(succ 28 29)
(succ 29 30)
(succ 30 31)
(succ 31 32)
(succ 32 33)
(succ 33 34)
(succ 34 35)
```

```
(succ 35 36)
(succ 36 37)
(succ 37 38)
(succ 38 39)
(succ 39 40)
(succ 40 41)
(succ 41 42)
(succ 42 43)
(succ 43 44)
(succ 44 45)
(succ 45 46)
(succ 46 47)
(succ 47 48)
(succ 48 49)
(succ 49 50)

(timesFive 0 0)
(timesFive 1 5)
(timesFive 2 10)
(timesFive 3 15)
(timesFive 4 20)
(timesFive 5 25)
(timesFive 6 30)
(timesFive 7 35)
(timesFive 8 40)
(timesFive 9 45)
(timesFive 10 50)
(timesFive 11 55)
(timesFive 12 60)
(timesFive 13 65)
(timesFive 14 70)
(timesFive 15 75)
(timesFive 16 80)
(timesFive 17 85)
(timesFive 18 90)
(timesFive 19 95)
(timesFive 20 100)
(timesFive 21 105)
(timesFive 22 110)
(timesFive 23 115)
(timesFive 24 120)
(timesFive 25 125)
(timesFive 26 130)
(timesFive 27 135)
(timesFive 28 140)
(timesFive 29 145)
(timesFive 30 150)
```

References

1. R Axelrod and WD Hamilton. The evolution of cooperation. *Science*, 211(4489):1390–1396, 1981.
2. Reyhan Aydoğan, Ivan Marsa-Maestre, Mark Klein, and Catholijn M Jonker. A machine learning approach for mechanism selection in complex negotiations. *Journal of Systems Science and Systems Engineering*, 27(2):134–155, 2018.
3. Tim Baarslag, Koen Hindriks, Mark Hendrikx, Alexander Dirkzwager, and Catholijn Jonker. Decoupling negotiating agents to explore the space of negotiation strategies. In Ivan Marsa-Maestre, Miguel A. Lopez-Carmona, Takayuki Ito, Minjie Zhang, Quan Bai, and Katsuhide Fujita, editors, *Novel Insights in Agent-based Complex Automated Negotiation*, pages 61–83. Springer Japan, Tokyo, 2014.
4. Tim Baarslag, Koen Hindriks, Catholijn M. Jonker, Sarit Kraus, and Raz Lin. The first automated negotiating agents competition (ANAC 2010). In Takayuki Ito, Minjie Zhang, Valentin Robu, Shaheen Fatima,

and Tokuro Matsuo, editors, *New Trends in Agent-based Complex Automated Negotiations, Series of Studies in Computational Intelligence*. Springer-Verlag, 2010.

5. Tristan Cazenave and Abdallah Saffidine. Score bounded monte-carlo tree search. In *Computers and Games*, volume 6515 of *Lecture Notes in Computer Science*, pages 93–104. Springer, 2010.
6. Siqi Chen and Gerhard Weiss. An efficient automated negotiation strategy for complex environments. *Eng. Appl. of AI*, 26(10):2613–2623, 2013.
7. Armin Chitizadeh and Michael Thielscher. General language evolution in general game playing. In Tanja Mitrovic, Bing Xue, and Xiaodong Li, editors, *AI 2018: Advances in Artificial Intelligence*, pages 51–64, Cham, 2018. Springer International Publishing.
8. Dave de Jonge, Tomas Trescak, Carles Sierra, Simeon Simoff, and Ramon López de Mántaras. Using game description language for mediated dispute resolution. *AI & SOCIETY*, 34(4):767–784, Dec 2019.
9. Enrique de la Hoz, Ivan Marsá-Maestre, José Manuel Giménez-Guzmán, David Orden, and Mark Klein. Multi-agent nonlinear negotiation for wi-fi channel assignment. In Kate Larson, Michael Winikoff, Sanmay Das, and Edmund H. Durfee, editors, *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, pages 1035–1043. ACM, 2017.
10. Harmen de Weerd, Rineke Verbrugge, and Bart Verheij. Higher-order theory of mind in negotiations under incomplete information. In Guido Boella, Edith Elkind, Bastin Tony Roy Savarimuthu, Frank Dignum, and Martin K. Purvis, editors, *PRIMA 2013: Principles and Practice of Multi-Agent Systems - 16th International Conference, Dunedin, New Zealand, December 1-6, 2013. Proceedings*, volume 8291 of *Lecture Notes in Computer Science*, pages 101–116. Springer, 2013.
11. Angela Fabregues. *Facing the Challenge of Automated Negotiations with Humans*. PhD thesis, Universitat Autònoma de Barcelona, 2012.
12. Angela Fabregues and Carles Sierra. Dipgame: a challenging negotiation testbed. *Engineering Applications of Artificial Intelligence*, 2011.
13. Peyman Faratin, Carles Sierra, and Nicholas R. Jennings. Negotiation decision functions for autonomous agents. *Robotics and Autonomous Systems*, 24(3-4):159 – 182, 1998. Multi-Agent Rationality.
14. Peyman Faratin, Carles Sierra, and Nicholas R. Jennings. Using similarity criteria to make negotiation trade-offs. In *International Conference on Multi-Agent Systems, ICMAS'00*, pages 119–126, 2000.
15. André Ferreira, Henrique Lopes Cardoso, and Luís 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.
16. Sevan G. Ficici and Avi Pfeffer. Modeling how humans reason about others with partial information. In Lin Padgham, David C. Parkes, Jörg P. Müller, and Simon Parsons, editors, *7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Estoril, Portugal, May 12-16, 2008, Volume 1*, pages 315–322. IFAAMAS, 2008.
17. Hilmar Finnsson. *Simulation-Based General Game Playing*. PhD thesis, School of Computer Science, Reykjavik University, 2012.
18. Jose M Font, Tobias Mahlmann, Daniel Manrique, and Julian Togelius. A card game description language. In *European Conference on the Applications of Evolutionary Computation*, pages 254–263. Springer, 2013.
19. Katsuhide Fujita, Reyhan Aydoğan, Tim Baarslag, Takayuki Ito, and Catholijn M. Jonker. The fifth automated negotiating agents competition (ANAC 2014). In *Recent Advances in Agent-based Complex Automated Negotiation .*, volume 638 of *Studies in Computational Intelligence*, pages 211–224. Springer, 2014.
20. Y. Gal, B. Grosz, S. Kraus, A. Pfeffer, and S. Shieber. Agent decision-making in open-mixed networks. *Artificial Intelligence*, 2010.
21. Sylvain Gelly and David Silver. Combining online and offline knowledge in uct. In *Proceedings of the 24th international conference on Machine learning*, pages 273–280. ACM, 2007.
22. M. Genesereth, N. Love, and B. Pell. General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2):62–72, 2005.
23. Barbara J. Grosz, Sarit Kraus, Shavit Talman, Boaz Stossel, and Moti Havlin. The influence of social dependencies on decision-making: Initial investigations with a new game. In *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), 19-23 August 2004, New York, NY, USA*, pages 782–789. IEEE Computer Society, 2004.
24. Litan Ilany and Ya'akov Gal. Algorithm selection in bilateral negotiation. *Autonomous Agents and Multi-Agent Systems*, 30(4):697–723, 2016.
25. Takayuki Ito, Mark Klein, and Hiromitsu Hattori. A multi-issue negotiation protocol among agents with nonlinear utility functions. *Multiagent Grid Syst.*, 4:67–83, January 2008.
26. Guifei Jiang, Dongmo Zhang, Laurent Perrussel, and Heng Zhang. Epistemic gdl: A logic for representing and reasoning about imperfect information games. In *IJCAI*, volume 16, pages 1138–1144, 2016.

27. Dave de Jonge, Tim Baarslag, Reyhan Aydoğan, Catholijn Jonker, Katsuhide Fujita, and Takayuki Ito. The challenge of negotiation in the game of diplomacy. In Marin Lujak, editor, *Agreement Technologies 2018, Revised Selected Papers*, pages 100–114, Cham, 2019. Springer International Publishing.
28. Dave de Jonge and Carles 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.
29. Dave de Jonge and Carles Sierra. GANGSTER: an automated negotiator applying genetic algorithms. In Naoki Fukuta, Takayuki Ito, Minjie Zhang, Katsuhide Fujita, and Valentin Robu, editors, *Recent Advances in Agent-based Complex Automated Negotiation*, Studies in Computational Intelligence, pages 225–234. Springer International Publishing, 2016.
30. Dave de Jonge and Carles Sierra. D-Brane: a diplomacy playing agent for automated negotiations research. *Applied Intelligence*, 47(1):158–177, 2017.
31. Dave de Jonge and Dongmo Zhang. Using gdl to represent domain knowledge for automated negotiations. In Nardine Osman and Carles Sierra, editors, *Autonomous Agents and Multiagent Systems: AAMAS 2016 Workshops, Visionary Papers, Singapore, Singapore, May 9-10, 2016, Revised Selected Papers*, pages 134–153. Springer International Publishing, Cham, 2016.
32. Dave de Jonge and Dongmo Zhang. Automated negotiations for general game playing. In Kate Larson, Michael Winikoff, Sanmay Das, and Edmund Durfee, editors, *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, pages 371–379. ACM, 2017.
33. Dave de Jonge and Dongmo Zhang. Strategic negotiations for extensive-form games. *Autonomous Agents and Multi-Agent Systems*, 34(1), Apr 2020.
34. Donald E. Knuth and Ronald W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293 – 326, 1975.
35. Levente Kocsis and Csaba 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.
36. S. Kraus, D. Lehman, and E. Ephrati. 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.
37. Sarit Kraus and Daniel Lehmann. Designing and building a negotiating automated agent. *Computational Intelligence*, 11:132–171, 1995.
38. Raz Lin, Sarit Kraus, Tim Baarslag, Dmytro Tykhonov, Koen Hindriks, and Catholijn M. Jonker. Genius: An integrated environment for supporting the design of generic automated negotiators. *Computational Intelligence*, 30(1):48–70, 2014.
39. Nathaniel Love, Michael Genesereth, and Timothy 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.
40. Yunbo Lv, Junwu Zhu, and Yi Jiang. Using egdl to represent domain knowledge for imperfect information automated negotiations. *Journal of Ambient Intelligence and Humanized Computing*, 2020.
41. João Marinheiro and Henrique Lopes Cardoso. Towards general cooperative game playing. In Ngoc Thanh Nguyen, Ryszard Kowalczyk, Jaap van den Herik, Ana Paula Rocha, and Joaquim Filipe, editors, *Transactions on Computational Collective Intelligence XXVIII*, pages 164–192, Cham, 2018. Springer International Publishing.
42. Ivan Marsa-Maestre, Miguel A. Lopez-Carmona, Juan R. Velasco, and Enrique 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.
43. Ivan Marsa-Maestre, Miguel A. Lopez-Carmona, Juan R. Velasco, Takayuki Ito, Mark Klein, and Katsuhide 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.
44. Johnathan Mell, Gale M. Lucas, and Jonathan Gratch. An effective conversation tactic for creating value over repeated negotiations. In *AAMAS*, pages 1567–1576. ACM, 2015.
45. J.F. Nash. The bargaining problem. *"Econometrica"*, "18":155–162, 1950.
46. Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Ann. Math. Artif. Intell.*, 25(3-4):241–273, 1999.
47. Makoto Niimi and Takayuki Ito. Agentm. In Naoki Fukuta, Takayuki Ito, Minjie Zhang, Katsuhide Fujita, and Valentin Robu, editors, *Recent Advances in Agent-based Complex Automated Negotiation [revised and extended papers from the 7th International Workshop on Agent-based Complex Automated Negotiation, ACAN 2014, Paris, France, May 2014].*, volume 638 of *Studies in Computational Intelligence*, pages 235–240. Springer, 2014.

48. Martin J. Osborne and Ariel Rubinstein. *Bargaining and Markets*. Academic Press, 1990.
49. M.J. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, 1994.
50. Noam Peled, Sarit Kraus, et al. A study of computational and human strategies in revelation games. *Autonomous Agents and Multi-Agent Systems*, 29(1):73–97, 2015.
51. J. S. Rosenschein and G. Zlotkin. *Rules of Encounter*. The MIT Press, Cambridge, USA, 1994.
52. Robert W Rosenthal. Games of perfect information, predatory pricing and the chain-store paradox. *Journal of Economic Theory*, 25(1):92 – 100, 1981.
53. Víctor Sánchez-Anguix, Reyhan Aydogan, Vicente Julián, and Catholijn M. Jonker. Intra-team strategies for teams negotiating against competitor, matchers, and conceders. In *Novel Insights in Agent-based Complex Automated Negotiation*, volume 535 of *Studies in Computational Intelligence*, pages 3–22. Springer, 2014.
54. Tian Sang and Jiun-Hung Chen. Beyond minimax: Nonzero-sum game tree search with knowledge oriented players. In *Metareasoning: Thinking about Thinking, Papers from the AAAI Workshop, Technical Report WS-08-07*. AAAI Press, Menlo Park, California, 2008.
55. Tom Schaul. A video game description language for model-based or interactive learning. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, pages 1–8. IEEE, 2013.
56. Stephan Schiffel and Michael 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.
57. Stephan Schiffel and Michael Thielscher. A multiagent semantics for the game description language. In *International Conference on Agents and Artificial Intelligence*, pages 44–55. Springer, 2009.
58. Eric Schkufza, Nathaniel Love, and Michael Genesereth. Propositional automata and cell automata: Representational frameworks for discrete dynamic systems. In Wayne Wobcke and Mengjie Zhang, editors, *AI 2008: Advances in Artificial Intelligence*, pages 56–66, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
59. Roberto Serrano. bargaining. In Steven N. Durlauf and Lawrence E. Blume, editors, *The New Palgrave Dictionary of Economics*. Palgrave Macmillan, Basingstoke, 2008.
60. Martin Shubik. The dollar auction game: A paradox in noncooperative behavior and escalation. *The Journal of Conflict Resolution*, 15(1):109–111, 1971.
61. David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.
62. Michael 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.
63. Michael Thielscher. The general game playing description language is universal. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.
64. Michael Thielscher. Gdl-iii: A description language for epistemic general game playing. In *The IJCAI-16 Workshop on General Game Playing*, page 31, 2017.
65. Robert Shaun Thomas. Real-time decision making for adversarial environments using a plan-based heuristic, 2003.
66. John von Neumann. On the theory of games of strategy. In A.W. Tucker and R.D. Luce, editors, *Contributions to the Theory of Games*, pages 13–42. Princeton University Press, 1959.
67. Natalia Voynarovskaya, Roman Gorbunov, Emilia Barakova, Rene Ahn, and Matthias Rauterberg. Nonverbal behavior observation: Collaborative gaming method for prediction of conflicts during long-term missions. In *International Conference on Entertainment Computing*, pages 103–114. Springer, 2010.
68. Colin R. Williams, Valentin Robu, Enrico H. Gerding, and Nicholas R. Jennings. Using gaussian processes to optimise concession in complex negotiations against unknown opponents. In *IJCAI*, pages 432–438. IJCAI/AAAI, 2011.
69. Konstantia Xenou, Georgios Chalkiadakis, and Stergos D. Afantenos. Deep reinforcement learning in strategic board game environments. In Marija Slavkovik, editor, *Multi-Agent Systems - 16th European Conference, EUMAS 2018, Bergen, Norway, December 6-7, 2018, Revised Selected Papers*, volume 11450 of *Lecture Notes in Computer Science*, pages 233–248. Springer, 2018.
70. Dongmo Zhang and Michael 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.