Artificial Intelligence Research and Development
M. Villaret et al. (Eds.)

doi:10.3233/FAIA210120

Enabling Game-Theoretical Analysis of Social Rules

Nieves MONTES [a,1], Nardine OSMAN [a] and Carles SIERRA [a]
[a] *Artificial Intelligence Research Institute (IIIA-CSIC)*
Campus de la UAB, 08193 Bellaterra (Barcelona)

Abstract. In the field of normative multiagent systems, the relationship between a game structure and its underpinning agent interaction rules is hardly ever addressed in a systematic manner. In this work, we introduce the Action Situation Language (ASL), inspired by Elinor Ostrom's Institutional Analysis and Development framework, to bridge the gap between games and rules. The ASL provides a syntax for the description of agent interactions, and is complemented by an engine that automatically provides semantics for them as extensive-form games. The resulting games can then be analysed using standard game-theoretical solution concepts, hence allowing any community of agents to automatically perform *what-if* analysis of potential new interaction rules.

Keywords. normative multiagent systems, game theory, rules, logic programming, Institutional Analysis and Development framework

1. Introduction

In the field of normative multiagent systems (norMAS), a great deal of work has been devoted to the study of *norms*, *rules* and other mechanisms to achieve coordination among autonomous agents [1,2,3]. In parallel, game theory has provided a powerful toolbox to model multiagent interactions of competitive, cooperative and hybrid nature. Very well established game theoretical solution concepts are prevalent across the Multiagent Systems literature (e.g. [4,5]). However, in game theory, the rules that configure the structure of the interaction become irrelevant once the formal model has been built, and are often expressed in non-systematic, plain natural language.

The fundamental contribution of this paper is a formal methodology for the *what-if* analysis of community rules through the game structures they generate. To do so, we define the syntax of the novel Action Situation Language (ASL), inspired by the Institutional Analysis and Development (IAD) framework. We formally define its semantics as an extensive-form game (EFG) and provide an engine to automatically build it from an ASL description, hence connecting the norMAS and game theory fields. The choice of EFGs as the ASL semantics is motivated by the availability of many reasoning schemes from the game theory literature. The application of these schemes to the resulting model completes the pipeline from a rule specification to an evaluation of the outcomes it promotes.

[1]Contact: {nmontes,nardine,sierra}@iiia.csic.es

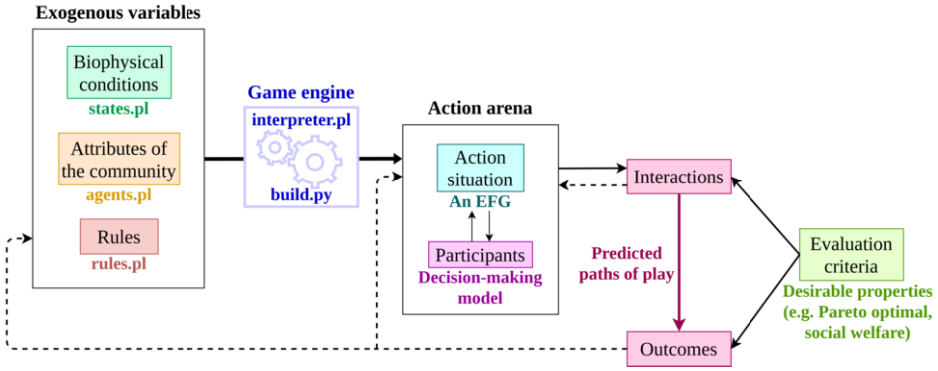

Figure 1. Outline of the IAD framework (adapted from [6, p. 15]). Coloured text outside boxes indicate either the script that contain the information on the boxed component, or the game-theoretical concepts that would represent it.

In the remainder of this paper, we provide some background on the IAD framework (Section 1.1) and review some research efforts similar to ours (Section 1.2). Then, we move on to the main part and present the syntax and semantics of ASL in Sections 2 and 3 respectively. We complement those two with a running example. Finally, we conclude and point to possible future directions in Section 4.

1.1. Background

Within the field of policy analysis, the Institutional Analysis and Development (IAD) framework, put forward by Ostrom and colleagues [6], represents a comprehensive theoretical effort to identify and delineate the universal building blocks that make up any social interaction. Its main components are presented in Figure 1. In the central part, the social interaction under study (e.g. a commercial transaction, a legislature) is referred to as an *action arena*. It is made up of a set of *participants* (endowed with some decision-making model) who find themselves within an *action situation*, which is broadly defined as the social space they might enter, take actions in and jointly bring about outcomes.

Action arenas are conditioned by three sets of exogenous variables: biophysical conditions, attributes of the community, and rules. The first two refer to environmental and material features. Within the scope of this work, the term *rules* will encapsulate both the laws of nature that inevitably play a part in determining outcomes, as well as malleable human-made regulations that constrain or provide alternative avenues for a course of action. Physical laws are distinct from biophysical conditions in the sense that laws control the dynamics of the environment (drop an object and it will land on the ground) while biophysical conditions refer to static elements (like land topology). Meanwhile, human-made regulative rules play an essential part when it comes to achieving more positive outcomes, as measured by the evaluation criteria of choice. Suitable changes to the regulatory rules have the potential to steer the interaction towards more desirable endings, by modifying the incentives that agents face.

Within an action situation, the IAD framework also delineates seven distinct internal variables that compose any social space. Of these, we will focus on the four that we deem indispensable: (1) the participants who are allowed to enter the interaction; (2) the

roles they take on; (3) the actions assigned to every role; and (4) the linkages between the executed actions and the outcomes they bring about. Every one of these components will have a dedicated rule type in our language.

1.2. Related work

Originally, the IAD framework was complemented by the Institutional Grammar (IG) syntax [7], which parses institutional statements into several distinct fields. Lately, the IG has spurred renewed interest, with extensions to the original proposal including the nesting of statements [8] and the distinction between different levels of granularity in the parsing [9]. Although the early version of IG did include the derivation of some game-theoretical analysis from a set of statements [6, Ch. 5-6], no attempt is made to automate this process, as the IG syntax is not formulated as a *machine-readable* language. Some works that attempt to make it operational are limited in their scope [10,11], as they only use statements to encode agent strategies in an evolutionary type simulation.

On another front, the field of General Game Playing within the AI community has come up throughout the years with machine-processable languages for the specification of games. The most prominent of these is the Game Description Language (GDL) [12] and its extensions to imperfect-information [13] and epistemic games [14]. Beyond game playing, GDL has been used for more socially relevant applications, such as mediated dispute resolution [15] and automated negotiations [16].

The original GDL admits a form of restricted imperfect information as simultaneous moves that we incorporate into our language. However, the rules of the game are implicit in GDL descriptions, while our language represents them explicitly and individually (one by one). Therefore, ASL descriptions are more declarative than GDL ones, as new rules can be easily added and their individual impact examined.

Also within the AI community, declarative action representations are ubiquitous in the planning domain. The multiagent extension to the Planning Domain Definition Language (MA-PDDL) [17] and ASL have similar expressive power for actions: both allow the specification of concurrent actions with probabilistic effects. A difference between the two representations is the removal of state fluents: MA-PDDL specifies them explicitly within action effects, while ASL relies on incompatibilities between the previous state fluents and the newly derived ones to remove outdated facts.

2. ASL syntax

Now, we turn to the definition of the syntax of our Action Situation Language (ASL). To do so, we leverage the conceptual clarity of the IAD framework and tailor the design of our language to the components delineated in that theory. ASL is a logical language implemented in Prolog, hence fully machine-readable yet relatively syntactically friendly. In order to fully specify a complete action situation, it should, first, include constructs for the three sets of exogenous variables that determine it (see Figure 1):

- The candidate agents to take part in the interaction, plus any relevant characteristics (**attributes of the community**): age, gender, ethnicity, etc.
- The **biophysical and environmental conditions**, like land topology, location of resources, etc.

Table 1. Action Situation Language keywords, sorted into reserved predicate symbols (with their arity) and operators (with their type in parenthesis).

Predicates			Operators	
`agent/1`	`rule/4`	`initially/1`	`if` (prefix)	`then` (infix)
`participates/1`	`role/2`	`incompatible/2`	`where` (infix)	$\sim$ (prefix)
`can/2`	`does/2`	`terminal/0`	`withProb` (infix)	`and` (infix)

- The **rules** structuring the interaction, in particular the following four rule *types*:
  - **Boundary rules** determine who is allowed to participate in the interaction.
  - **Position rules** assign (possibly several) roles to participants.
  - **Choice rules** establish the actions available to every role under the current circumstances.
  - **Control rules** relate (possibly joint) actions to the effects they have.

Additionally, the following is also necessary:

- The starting point of the interaction, and the conditions under which it halts.
- Which facts describing a state are compatible with one another and can be simultaneously true (e.g., an individual cannot be at two different places at the same time).

The predicates for ASL are gathered in Table 1. Most of these appear as part of `rule` arguments, and only `agent`, `initially`, `terminal` and `incompatible` are used as standalone predicates.

We start by reviewing the predicate symbols that do not appear within rules. First, `agent(A)` simply designates `A` as an individual susceptible of entering the action situation. Second, `initially(F)` indicates that fact `F` holds true at the start of the interaction, prior to any action being executed. `terminal` plays the opposite role, as it returns true whenever the conditions for halting the interaction are met. Finally, `incompatible(F,L)` states that fact `F` cannot be simultaneously true with the fluents in list `L`.

We move on now to the syntax of rules. All rule clauses, regardless of the component they target, follow the general template in Figure 2. Their first argument is an identifier for the action situation where they apply. The second argument is their type. Third, the priority is a non-negative integer that determines which rule is to prevail in case several clauses have contradicting effects. Rules that model the unregulated situation[2] are assigned priority equal to zero, and are referred to as the *default* rules. The overwriting operator $\sim$ is introduced to have high priority rules nullify the effects of lower priority rules. The fourth and last argument of a `rule` predicate contains its content expressed as an *if-then-where* construct. The content of the `Condition` and `Consequence`

[2]By "unregulated", we mean that only rule statements that reflect physical principles are considered.

```
    Rule ::= rule(Id,Type,Priority,
                  if Condition then Consequence where Constraints).
    Type ::= boundary | position | choice | control
Priority ::= 0 | 1 | ... | ∞
```

Figure 2. General syntax of *if-then-where* rules.

Table 2. Syntactic restrictions for the `Condition` and `Consequence` fields for every of the proposed rule types. α stands for an atom, i.e. a predicate symbol with terms as arguments.

Rule type	Condition	Consequence
Boundary	`agent(Ag)`	[∼]`participates(Ag)`
Position	`participates(Ag)`	[∼]`role(Ag,R)`
Choice	`role(Ag,R)`	[∼]`can(Ag,Ac)`
Control	`joint_action`	[`consequence`$_1$ withProb p_1, `consequence`$_2$ withProb p_2, ...]
`joint_action` ::= `does(Ag,Ac)` [and `joint_action`] `consequence` ::= α [and `consequence`]		

fields is determined by the rule type in question. These restrictions are summarised in Table 2. `Constraints` always consists of a list of literals whose free variables unify with those in `Condition` and `Consequence`. The separation of rule pre-conditions into a short `Condition` and a `Constraints` field is not technically indispensable, but rather a stylistic choice to help keep the syntax concise.

Note that, in Table 2, boundary, position and choice rules have an analogous syntax: one `agent`, `participates` or `role` predicate as the `Condition`, and `participates`, `role` or `can` as the `Consequence`, respectively. In contrast, the control rules may have in their condition multiple `does` predicates concatenated by the `and` operator to reflect the execution of joint actions. Their consequences, instead of a single predicate, consists of a list where each of its members consists of predicates concatenated with `and`, and the whole conjunction is assigned some probability with the operator `withProb`. In order for a control rule to be valid, the probability distribution over the potential consequences must be well-defined, i.e. all p_i must fall in the range $[0, 1]$ and must add up to unity.[3]

Fishers example (syntax) The best way to understand the syntax of ASL is to provide a complete example of an action situation description.[4] Here, we present the model of an open fishery, where two fishers compete for two fishing spots (it is assumed that one is more productive than the other) [18, Ch. 4].[5] The clauses are split into three files according to the three exogenous variables identified in the IAD framework. First, agents.pl contains the information on the attributes of the community. It declares two agents and two attributes for each, `strength` and `speed` (the second one will become relevant later on when we introduce higher priority rules).

Second, states.pl introduces the environmental conditions. Here, two fishing spots are declared. This file also contains the `initially` and `terminal` clauses. All fishers start at the shore. The interaction halts when both fishers are at distinct spots (first `terminal` clause) or when one of them has lost a fight (second `terminal` clause). The last piece of information in states.pl is the `incompatible` clauses. They indicate that a fisher can only be at one location at a time, and that only one of them may be the winner of a fight or race.

Third, rules.pl contains the rule base. It only contains the *default* rules with priority equal to zero. All of them use the identifier "fishers". The first two rule statements are

[3] If that is not the case, the game engine (see Figure 1) will raise an error.

[4] Further examples can be found at the extended pre-print version of this paper, see: https://arxiv.org/abs/2105.13151.

[5] The complete ASL description for the fishers domain appears in Listing 6 (Appendix C) of the extended pre-print.

very generic boundary and position rules. They let all agents enter the action situation and denote all of them as fishers. Then, the choice rules indicate that fishers may go to any fishing spot from the shore and that, once at a spot, they may stay or leave for the other one.

The control rules regulate the effects that fishers' actions have on the environment. The first two control rules are related to the movement of fishers from the shore to a spot and between spots. Both of them are stated in terms of a single individual action and have deterministic effects (boats never break down). The last control rule does include joint actions and stochastic consequences. It states that when two different fishers who are at the same spot and take the same action will inevitably fight for the spot they meet at for the second time. The probability of each fisher winning the fight is proportional to their relative strength.

3. ASL semantics

As earlier introduced, an action situation description in ASL has its formal semantics grounded as an extensive-form game (EFG). This choice is motivated by the availability of well-established solution concepts within the game theory literature, which allow agents to reason on top of the resulting game. The construction of an EFG from an ASL description is composed of three main steps:[6] rule interpretation, game round building, and game round concatenation.

3.1. Rule interpretation

First, rule interpretation consists of querying the knowledge base to find, given the current state of the system, instantiations of the active rules (bindings to free variables), and processing their consequences. This task is performed by the interpreter.pl script (see Figure 1). Two groups of rules are distinguished regarding the processing of consequences.

On one hand, the common and relatively simple syntactic structure of boundary, position and choice rules make the processing of their consequences much easier. Essentially, what it amount to is the deletion of any fluent f if that same fluent preceded by the overwriting operator, $\sim f$, is derived from a higher priority rule. On the other hand, control rules need a much more thorough processing of consequences. Given a pre-transition state s_t (aka a set of fluents that completely characterise the current circumstances) and a joint action profile $\mu = \{\texttt{does}(ag_1, ac_1), \texttt{does}(ag_2, ac_2), ...\}$, the interpretation of control rules returns a set of potential post-transition states $S_{t+1} = \{s^1_{t+1}, s^2_{t+1}, ...\}$ and a probability distribution over those $P : S_{t+1} \to [0, 1]$.

We do not go into the details of this derivation and refer to the extended report for a detailed exposition. However, it is worth explaining how the interpretation of control rules tackles the *frame problem* [19]. This is an issue that any action formalism has to address. It states that, when the effects of an action are axiomatised, it should only be necessary to state the facts that do change due to it. Listing all variables that are not affected by a particular action is not to be required. This is precisely the case in ASL,

[6] This paper presents only an overview of the game building process. For a more detailed report, see Sections 3 and 4 of the extended pre-print.

function BUILD-GAME-ROUND(s_t):
Interpret choice rules to get the available action for every agent at s_t
Starting from the root node (identified with s_t), add an information set for every agent
For every terminal node z:
  Get the joint action profile μ executed to get from the root to z
  Interpret the control rules to get the potential next states S_{t+1} and probabilities P
  If there are stochastic effects ($\mid S_{t+1} \mid > 1$):
    Turn z into a chance node and add one child for every $s_{t+1} \in S_{t+1}$
    Set the edge probability to $P(s_{t+1})$
  Else identify z with s_{t+1} (the only element in S_{t+1})

Algorithm 1. BUILD-GAME-ROUNDS constructs the EFG that represents the execution of one action for agent in state s_t.

as control rules state in their `Consequence` field only the terms that do change. Then, prior to returning the set of potential next states S_{t+1}, the rule interpreter goes through the fluents in the pre-transition state s_t and, by performing queries to the `incompatible` clauses, determines which facts may be carried over to the post-transition states. Hence, fluents that are not affected by the actions in μ remain part of the state description.

3.2. Game round building and concatenation

The construction of the action situation semantics is performed by the script build.py (see Figure 1), which repeatedly communicates with the rule interpreter to get the processed consequences of rules.[7] In order to build the complete EFG semantics of an ASL description, the process is divided into the consecutive constructions of *game rounds*:

Definition 1. A *game round* is an extensive-form game[8] with the following characteristics:

- The root node is never a chance node.
- There is, at most, one information set[9] per player.
- For any two nodes x_1, x_2 that belong to the same information set, the length of the path from the root to x_1 and from the root to x_2 must be equal.
- If node x is a chance node, then all of its children are terminal.

In practice, a game round is an EFG where every agent has the opportunity to make at most one move. With imperfect information, the moves by every player are modelled as simultaneous. In this work, we use game rounds to model all the ways by which the system may transition from state s_t (the root of the game round) to a post-transition state in S_{t+1} (the terminal nodes) by executing any of the actions available at s_t according to the choice rules. We choose to use imperfect-information EFGs instead of normal-form games (the benchmark models for joint actions) because, through the use of chance

[7] The communication between Prolog and Python is achieved thanks to the open-source PySwip package: https://github.com/yuce/pyswip.

[8] For a thorough definition of EFGs, see [20].

[9] In an extensive-form game, an information set is a subset of a player's decision nodes such that, at the time of making a move, the player only knows that the system is in one of the subset's nodes, but not specifically which one.

function BUILD-FULL-GAME:
Interpret the boundary rules to get the set of participants
Interpret the position rules to get their roles
Set s_0 to the set of derivable instanced from `initially(F)`
$Q \leftarrow$ QUEUE(s_0)
While Q is not empty:
  $s_t \leftarrow$ POP(Q)
  If `?- terminal` returns `true` at s_t **then** continue
  $\gamma \leftarrow$ BUILD-GAME-ROUNDS(s_t)
  Append γ to overall game tree by s_t
  Push the terminal nodes in γ to Q

Algorithm 2. BUILD-FULL-GAME constructs the complete EFG semantics of an ASL description by concatenating game rounds.

nodes, EFGs explicitly store the information on the stochastic dynamics of the environment, a feature that is not available in normal-form games. Pseudo-code for the construction of game rounds is shown in Algorithm 1.

Now that we know how to build a single round, the only step that is left is their concatenation in order to build the complete game. Prior to that, the boundary and position rules have to be interpreted to get the participants and their roles, and the initial state of the system is derived as the set of instantiations of `initially(F)`. The pseudo-code for the function that concatenates the game rounds and builds the complete EFG semantics appears in Algorithm 2.

Note that, by construction, some of the nodes in the final game tree cannot be identified with the actual states of the system, but are auxiliary nodes necessary to capture the simultaneity of moves. Similarly, chance nodes do not correspond to actual states, but are needed to explicitly store the probabilities of random effects. In fact, the only nodes that can be identified with an actual state (i.e., with a set of fluents that completely characterise the circumstances of the system) are the root nodes of game rounds and the terminal nodes.

Typically, extensive-form games have some numerical rewards assigned to every agent at their leaf nodes. These quantities, typically referred to as the *utilities* of the game, serve as the objective function to implement various reasoning schemes. Our game building algorithms, however, do not assign utilities to the resulting leaf nodes. Once the complete game tree is constructed, we leave it to the discretion of the user to set rewards *a posteriori* (e.g., as a function of the fluents that hold at the terminal nodes and/or the path of play from the root node).

Fishers example: Semantics We complement the fishers action situation description with its corresponding game semantics, which appear in Figure 10 of the extended pre-print version. This extensive game has been built solely from the default rules, intended to capture the dynamics of the unregulated situation.

To illustrate the addition of a new policy, we append some extra rules to the action situation description with priority 1. This new extended description constitutes the *first-in-time, first-in-right* configuration. The additional rules are displayed in Listing 7 in Appendix C of the extended pre-print. Now, when agents leave the shore for the same fishing spot, they race to get there. The winner of the race is determined by the same mechanism as the loser of the fight was (by flipping a biased coin), but with the `speed`

Table 3. Terminal node and its associated state fluents that are most likely to be reached under the two rule configurations for the fishers action situation.

Rule configuration	Most likely outcome (Probability)
Default	15 - `at(alice,spot1),at(bob,spot1),won_fight(alice)` (0.31)
First-in-time, first-in-right	13 - `at(alice,spot2),at(bob,spot1),won_race(bob)` (0.62)

of the agents instead of their `strength`. This is captured by the last rule of type control. Then, the winner of the race is guaranteed the spot, meaning that he is obliged to stay, while the loser must leave. These requirements correspond to the two first rules of type choice. The resulting game semantics appear in Figure 11 of the extended pre-print version.

We set the utilities to the resulting game trees by assigning the following benefits and costs to some of the actions and outcomes: a fisher keeps a spot to himself or wins the fight over it ($v_1 = 10, v_2 = 5$), a fisher looses a fight ($d = -6$), a fisher travels between spots ($c = -2$). Then, we implement the computation of subgame perfect equilibrium (SPE) strategies, by computing the Nash equilibria at the final game rounds (following [21, p. 104]) and backtracking the expected utilities. In fact, the game semantics of ASL are particularly well suited for the implementation of subgame perfection rationality schemes, as every subgame corresponds to a combination of game rounds, and these are typically much smaller in size than the overall game tree, hence reducing computation requirements.

The most likely terminal nodes, and their associated fluents, predicted by the SPE strategies are displayed in Table 3. The default rule configuration predicts violence in the most likely outcome, whose probability is around 30%. In fact, this rule configuration leads to a violent outcome (leaf nodes 14 through 17, and 24 through 27) around 50% of the times the game is played. In contrast, the implementation of *first-in-time, first-in-right* rules avoid violence. By this evaluation criteria (avoidance of violence), the additional rules certainly lead to a more socially desirable outcome, thus the community may collectively agree to incorporate them.

4. Conclusions

In this work, we have defined the syntax and semantics of the Action Situation Language, which turns descriptions of social interactions into formal game models that can be later examined using the standard tools of game theory. Our contribution, coupled with some model of individual rationality and an evaluation criteria for the potential outcomes, amounts to a complete computational model of Ostrom's IAD framework.

The most interesting use that can be made of ASL is as a tool for the formal *what-if* analysis of community rules. The ability to introduce and retract rules into a single description is a feature that sets ASL apart from other game-oriented logical languages. We have illustrated such an analysis with an example of interest for policy analysts, the regulation of an open fishery through the introduction of *first-in-time, first-in-right* rules.

The work presented here can be expanded into several directions. For example, formal aspects of ASL, such as its integration with an action formalism (e.g. Situation Calculus), could be explored. On the more practical side, refinements to the language can also help enhance its expressive power. For example, a new type of information rules,

whose consequences deal with `sees` or `knows` predicates, could be introduced to regulate the observability of the current state, opening the door for extending the use of imperfect information beyond the modelling of simultaneous actions.

References

[1] Yoav Shoham and Moshe Tennenholtz. On social laws for artificial agent societies: off-line design. *Artificial Intelligence*, 73(1-2):231–252, 1995.

[2] Shmuel Onn and Moshe Tennenholtz. Determination of social laws for multi-agent mobilization. *Artificial Intelligence*, 95(1):155–167, aug 1997.

[3] Giulia Andrighetto, Guido Governatori, Pablo Noriega, and Leon van der Torre. Normative Multi-Agent Systems (Dagstuhl Seminar 12111). *Dagstuhl Reports*, 2(3):23–49, 2012.

[4] Carsten Hahn, Thomy Phan, Sebastian Feld, Christoph Roch, Fabian Ritz, Andreas Sedlmeier, Thomas Gabor, and Claudia Linnhoff-Popien. Nash equilibria in multi-agent swarms. In *Proceedings of the 12th International Conference on Agents and Artificial Intelligence*. SCITEPRESS - Science and Technology Publications, 2020.

[5] Philippe Caillou, Samir Aknine, and Suzanne Pinson. Searching pareto optimal solutions for the problem of forming and restructuring coalitions in multi-agent systems. *Group Decision and Negotiation*, 19(1):7–37, nov 2009.

[6] Elinor Ostrom. *Understanding Institutional Diversity*. Princeton University Press, September 2005.

[7] Sue E. S. Crawford and Elinor Ostrom. A grammar of institutions. *American Political Science Review*, 89(3):582–600, 1995.

[8] Christopher Frantz, Martin K. Purvis, Mariusz Nowostawski, and Bastin Tony Roy Savarimuthu. nADICO: A nested grammar of institutions. In *Lecture Notes in Computer Science*, pages 429–436. Springer Berlin Heidelberg, 2013.

[9] Christopher K. Frantz and Saba Siddiki. Institutional grammar 2.0: A specification for encoding and analyzing institutional design. *Public Administration*, 2021.

[10] Amineh Ghorbani and Giangiacomo Bravo. Managing the commons: a simple model of the emergence of institutions through collective action. *International Journal of the Commons*, 10(1):200–219, 2016.

[11] Alex Smajgl, Luis R. Izquierdo, and MArco Huigne. Modeling endogenous rule changes in an institutional context: the adico sequence. *Advances in Complex Systems*, 11(02):199–215, 2008.

[12] Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the aaai competition. *AI Magazine*, 26:62–72, 06 2005.

[13] S. Schiffel and M. Thielscher. Representing and reasoning about the rules of general games with imperfect information. *Journal of Artificial Intelligence Research*, 49:171–206, 2014.

[14] Michael Thielscher. Gdl-iii: A proposal to extend the game description language to general epistemic games. *Frontiers in Artificial Intelligence and Applications*, 285:1630–1631, 2016.

[15] 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, 2017.

[16] Dave de Jonge and Dongmo Zhang. GDL as a unifying domain description language for declarative automated negotiation. *Autonomous Agents and Multi-Agent Systems*, 35(1), 2021.

[17] Daniel L. Kovacs. A multi-agent extension of pddl3.1. In *Proceedings of the 3rd Workshop on the International Planning Competition (IPC), 22nd International Conference on Automated Planning and Scheduling (ICAPS-2012)*, pages 19–27. ICAPS, 2012.

[18] Elinor Ostrom, Roy Gardner, and Jimmy Walker. *Rules, Games, and Common-Pool Resources*. University of Michigan Press, 1994.

[19] Fangzhen Lin. *Situation Calculus*, volume 3 of *Foundations of Artificial Intelligence*, chapter 16, pages 649–669. Elsevier, 2008.

[20] Julio Díaz. *An introductory course on mathematical game theory*. American Mathematical Society and Real Sociedad Matemática Española, Providence, Rhode Island, USA and Madrid, 2010.

[21] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, October 2014.