Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI-23)
Journal Track

A Computational Model of Ostrom's Institutional Analysis and Development Framework (Extended Abstract)*

Nieves Montes , Nardine Osman , Carles Sierra
Artificial Intelligence Research Institute (IIIA-CSIC)
Campus de la UAB, Barcelona, Spain
{nmontes, nardine, sierra}@iiia.csic.es

Abstract

Ostrom's Institutional Analysis and Development (IAD) framework represents a comprehensive theoretical effort to identify and outline the variables that determine the outcome in any social interaction. Taking inspiration from it, we define the Action Situation Language (ASL), a machine-readable logical language to express the components of a multiagent interaction, with a special focus on the *rules* adopted by the community. The ASL is complemented by a game engine that takes an interaction description as input and automatically grounds its semantics as an Extensive-Form Game (EFG), which can be readily analysed using standard game-theoretical solution concepts. Overall, our model allows a community of agents to perform *what-if* analysis on a set of rules being considered for adoption, by automatically connecting rule configurations to the outcomes they incentivize.

1 Introduction

The Institutional Analysis and Development (IAD) framework (Figure 1) by Elinor Ostrom and colleagues is a theoretical framework that identifies the universal building blocks making up any social interaction [Ostrom, 2005]. There, any social encounter is referred to as an *action arena*, where a set of *participants* enter an *action situation*, which is the social space where they take actions and jointly achieve outcomes.

According to the IAD framework, there are three sets of exogenous variables that jointly determine the structure of an action situation. First, the *biophysical conditions* refer to features of the environment, such as land topology and location of resources. Second, the *attributes of the community* are variables intrinsically linked to the interacting agents, such as their age, gender, or ethnicity. Third, the *rules* encompass the statements that describe who can enter an action situation, which roles they can adopt, which actions they can execute and what are the consequences of those actions, among other elements. Rules are different from the other two variables in the sense that they are susceptible to change in the short term, if the agents decide that the current set of rules does not lead to positive outcomes.

In our work, we present a computational model of the IAD framework that allows a community of agents to automatically perform *what-if* analysis on the potential rules that they may be considering for adoption. Our contribution is twofold. First, we present the novel Action Situation Language (ASL). This is a machine-readable logical language (implemented in Prolog) whose syntax is highly tailored to the exogenous variables identified by the IAD. Using the ASL, formal and systematic description of social interactions can be written.

Our second contribution is a *game engine* that takes as input a valid ASL description and automatically builds its semantics as an Extensive-Form Game (EFG). These EFGs can then be analysed using standard game-theoretical solution concepts (such as the Nash equilibrium, although our work is agnostic with respect to the particular agent decision-making model). This analysis induces a distribution over the possible outcomes of the game (i.e. the terminal nodes in the game tree). Simultaneously, such outcomes can be evaluated according to some criteria of choice, such as optimality, efficiency, or some notion of social welfare. Thus, one can evaluate the game generated by a particular ASL description by computing the expected value of the evaluation criteria of choice, and decide whether it is satisfactory or alternative regulations should be explored.

Previous work in AI on General Game Playing (GGP) has proposed a number of languages for the specification of general games, most notably the Game Description Language (GDL) [Genesereth *et al.*, 2005] and its extensions GDL-II [Schiffel and Thielscher, 2014] and GDL-III [Thielscher, 2017]. Although GDL has been used for some socially-oriented applications [de Jonge *et al.*, 2017; de Jonge and Zhang, 2021], there are several features that make ASL much more suitable than GDL to represent socioeconomic interactions. First, ASL separates "rules" from the other exogenous variables, while GDL considers that the entire game description constitutes the "rules" of the game. Hence, ASL clearly specifies which aspect of the interaction configuration is susceptible to revision, while GDL does not. This leads to the second feature setting ASL apart from GDL, and that is that ASL descriptions are meant to be *extensible*, into which new rules can be appended. Potential conflicts that may arise between these new rules and existing ones are handled by the

*Full paper in Artificial Intelligence, Volume 311, 2022 [Montes *et al.*, 2022].

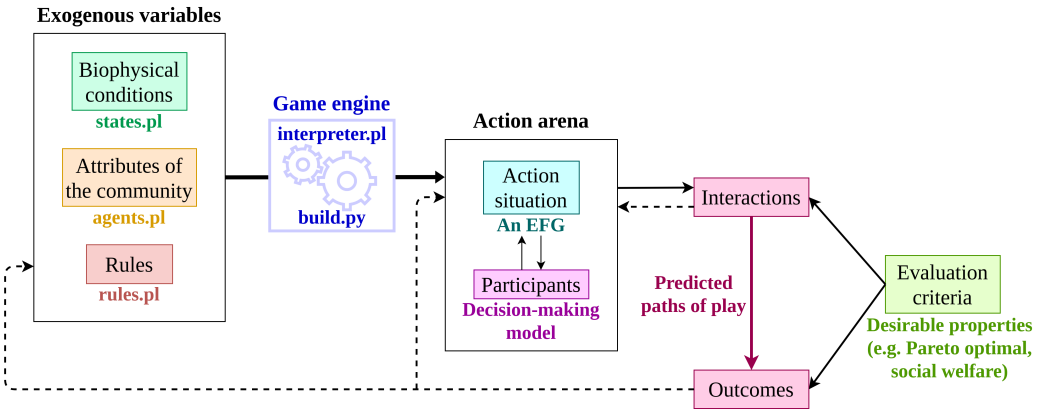


Figure 1: Outline of the computational model of the IAD framework.

game engine. In contrast, GDL descriptions are static and not meant for extension.

2 Syntax

ASL descriptions are split into three files (agents.pl, states.pl and rules.pl) corresponding to the three sets of exogenous variables in the IAD framework (see Figure 1).

First, agents.pl contains all the information pertaining to the attributes of the community that participates in the action situation. Here, agents are declared using a reserved predicate symbol `agent/1`, e.g. `agent(alice)`. Additional domain-dependent attributes can be encoded with facts `feature_name(Agent, Value)`, e.g. `age(alice,34)`.

Second, states.pl contains information pertaining to the biophysical conditions. Environmental features of the system can be declared using domain-dependent predicates. For example, in the fishers' examples presented later, the existence of a fishing spot is stated as `fishing_spot(spot1)`. Besides domain-dependent information, states.pl also includes facts with the reserved predicate `initially/1` to express the initial conditions of the system. For example, the clause `initially(at(Ag,shore)) :- role(Ag,fisher)` states that at the beginning of the interaction all fishers are on the shore. Finally, states.pl contains the information on what literals describing a state of the system are incompatible with one another, using the reserved predicate `incompatible/2`. Then, `incompatible(F,L)` states that fact `F` cannot be simultaneously true with the literals in list `L`.

Third, rules.pl contains all the rule statements that configure the social interaction. They are expressed through facts with reserved predicate `rule/4` that follow the syntax in Figure 2. The first argument of a rule is an identifier (`Id`) that indicates the action situation to which the rule pertains. The second argument is the *type* of the rule, and has possible values "boundary", "position", "choice" or "control". The type indicates which aspect of the action situation the rule is regulating. Boundary rules determine which agents are allowed to participate. Position rules determine which participants are allowed to take on which roles. Choice rules assign actions to roles. Last, control rules determine what is the effect on the state of the system that the execution of actions produce. The function of these 'types' is to indicate to the game engine at which stage of the game construction process the rules in question should be queried (see Section 3).

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

Figure 2: General syntax of `rule/4` statements.

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`]		

Table 1: Syntactic restrictions for the `Condition` and `Consequence` fields per rule type.

The third argument of `rule/4` statements is their `Priority`. This is a non-negative integer indicating which rule is to prevail in case several rules enter into conflict. Higher-priority rules take precedence over lower priority ones. We refer to the rules with priority equal to zero as the set of *default* rules. The fourth and last argument of `rule/4` statements is their content, expressed as an if-then-where statement. The `Condition` and `Consequence` fields are subject to further syntactical restrictions according to their rule type (Table 1).

Boundary, position and choice rules all have an analogous syntax: one `agent/1`, `participates/1` or `role/2` literal as their `Conditions`, and one `participates/1`, `role/2` or `can/2` literal as their `Consequence`, respectively. Also, their `Consequence` literal might be preceded by the overwriting operator ∼, which is used to have higher-

priority rules overwrite lower-priority ones. Thanks to their analogous syntactic structure, boundary, position and choice rules are all processed by the same function during the game construction process.

In contrast, control rules have a much richer syntax. Their `Condition` includes multiple `does/2` literals combined by operator `and`, to account for the possibility that some effects are only brought about by joint actions. Their `Consequence` is a list where each member consists of (possibly several) literals, also combined by the `and` operator. These literals are domain-specific, chosen to describe the relevant features of the interaction under examination.

The conjunction of literals that makes up for one potential consequence of a control rule is assigned some probability through the operator `withProb`. In this way, one can express non-deterministic consequences of actions in stochastic environments, as long as the probability distribution over the potential consequences is valid (i.e. it adds up to unity and all probabilities are between 0 and 1). Of course, deterministic environments can be simply described by having a single element in the `Consequences` list with an assigned probability of 1.

All rules, regardless of their type, have a `Constraints` field in their content. This consists of a list of literals with the free variables that unify with the variables in the `Condition` and `Consequence` of the rule.

Example. To illustrate the syntax of ASL, the reader is referred to the fishers example, available in the open-source repository of our IAD framework model.[1] This example first appeared as a handcrafted policy analysis exercise to exemplify how community-crafted rules are capable of transforming the opportunity structure in common-pool resource situations [Ostrom *et al.*, 1994]. Here, a community of fishers compete for fishing spots in an open-water fishery. Fishers may fight over spots to determine who will fish there, or they may come up with additional regulations that implement an allocation scheme that avoids violence while being as fair as possible to all fishers in the community.

In the agents.pl file, two agents: (`alice` and `bob`) are declared, alongside with their `strength` and `speed` attributes, which are graded on a scale from 0 to 10.

In the states.pl file, two fishing spots are declared as `spot1` and `spot2`. Next, it is stated that, initially, all agents with role `fisher` are at the shore. Termination conditions are met when different fishers are in different spots, or when a fight has occurred. Finally, the `incompatible/2` clauses state that an agent can only be in one location at a time, and that there is only one possible winner for fights or races.

In the default configuration (i.e. the rules with `Priority` equal to 0), the only boundary rule allows all declared agents to participate, while the only position rule assigns to all of them the role `fisher`. The choice rules provide fishers with the possibility to go to any fishing spot from the shore and, once there, decide to either stay or leave. Control rules ensure that if fishers go to a spot from the shore, or they switch one spot for another, their movement is always successful. However, if both fishers take the same action (either `stay` or `leave`) when they are both at the same spot, hence meeting again, they fight for that spot. The winner of the fight is chosen randomly according to a probability proportional to each agent's strength rating.

Other than default rules, this ASL description also includes higher-priority rules to implement alternative spot-allocation schemes that do not involve fighting. These are *first-in-time, first-in-right* rules (which assign a fishing spot to the fisher who arrived there first) and *first-to-announce, first-in-right* rules that randomly appoint a participant to be an *announcer*. The announcer can declare a spot and, as long as it leaves the shore directly for the announced spot, it can claim privileges over it. A detailed explanation of these higher-priority rules can be found in the full paper.

3 Rule Interpretation and Semantics

Once a valid ASL description has been written following the semantics of Section 2, a *game engine* (see Figure 1) is capable of automatically interpreting it and grounding its semantics as an EFG [González-Díaz *et al.*, 2010]. The game engine has two functions: the first is to query, interpret and process the consequences of rules in the input ASL description; the second is to assemble an EFG based on the results from the previous step.

The interpretation of rules is the most important task that the game engine handles. To perform it, it must first query the ASL description to find which rules currently apply and which are their grounded (i.e. without any free variables) consequences. For this step, the game engine relies on the fact that a rule statement of the form:

```
rule(..., if Condition then Consequence
where [Constraint1, Constrains2, ...]).
```

is equivalent to a classical Prolog clause of the form:

```
Consequence :- Condition, Constraint1,
          Constraint2, ...  .
```

The game engine does not query for all currently applicable rules at once. Rather, it queries for rules of a particular type, depending on the stage of the game construction process that it is in (as illustrated in the EFG construction steps presented shortly).

Once the currently active rules have been queried, the game engine processes all of their consequences. In particular, it handles conflicts between contradictory rules, based on their priorities and the `incompatible/2` statements. We refer to the full paper for a detailed description of the conflict-solving mechanism, where rules with higher priority deny the consequences of rules with lower priority.

The result of processing the consequences of currently active rules feeds into the game engine's other main task: to build the semantics of the ASL description as an EFG model. An overview of this procedure goes as follows:

1. Get the participants from the *boundary rules*, and add them to the ASL description as `participant(<Agent>).`
2. Assign participants to their roles using *position rules*, and add them to the ASL description as `role(<Agent>,<Role>).`

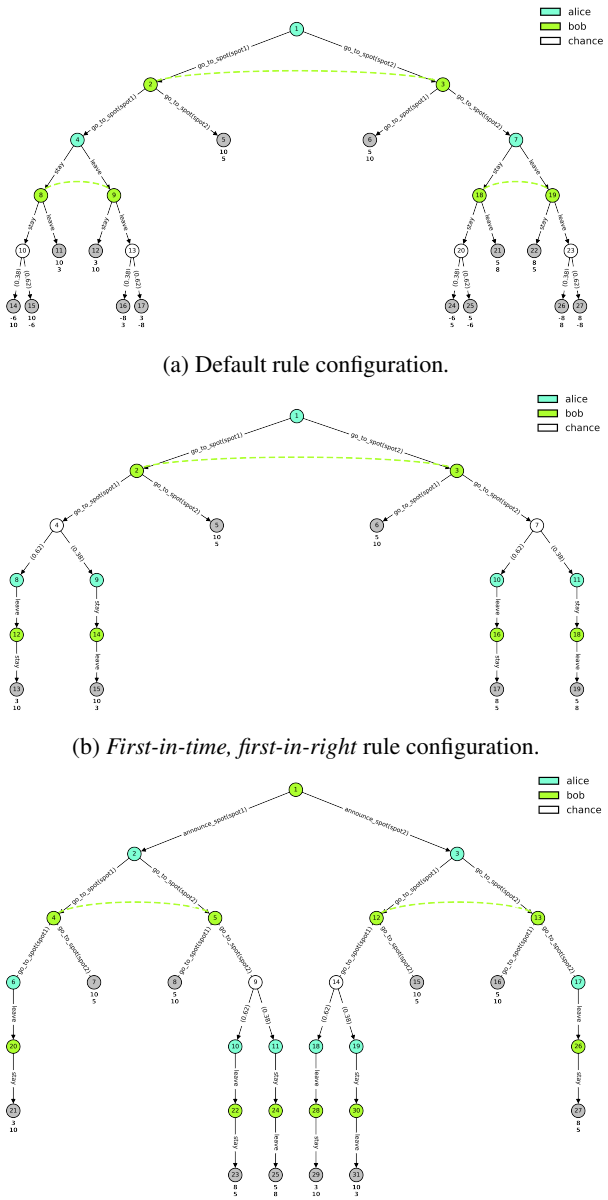
[1] shorturl.at/GLO07

3. Get the initial conditions using the `initially/2` statements. Start building the EFG by creating its root node and assigning the initially true facts to it.
4. While `terminal` does not return `true`:
   (a) Process the *choice rules* to find which actions are available to agents. Build a subtree emanating from the root node that reflects these possibilities (i.e. by adding new nodes, edges labelled with the available actions and information sets to reflect simultaneous moves).
   (b) For every new terminal node generated, query the *control rules* to find which facts are now true, provided that the corresponding joint actions from the root node have been executed. Assign these facts to the terminal node in question.
   (c) Select a new unexplored terminal node as the new root node for the next iteration.

Once the game model has been built, it can be analysed using standard game-theoretical solution concepts, such as the Nash equilibrium. This step is equivalent to the introduction of a decision-making model for the agents (i.e. the "Participants" box in Figure 1). Nevertheless, our model of the IAD framework is agnostic with respect to the specific decision-making model implemented.

The introduction of such decision-making models for agents induces a probability distribution over the possible terminal nodes of the game, i.e. the outcomes that the community of agents may achieve. Then, these outcomes can be evaluated according to some metric (i.e. the "Evaluation criteria" box in Figure 1), such as their efficiency, optimality, or some notion of social welfare. The expectation of this metric can be computed by weighting it by the probability distribution over possible outcomes, thus leading to a quantitative evaluation of the EFG and, by extension, of the ASL description that generated it. This evaluation can serve as a motivation for agents to introduce new higher-priority rules, if they deem that the current rule configuration does not satisfy their needs.

Example. The resulting game semantics built from the various rule configurations for the fishers example presented in Section 2 are presented in Figure 3. Besides different game tree topology, our analysis shows that, for the default rule configuration, violent outcomes are predicted to happen about 50% of the times that the interaction takes place. The introduction of *first-in-time, first-in-right* rules eliminates violent outcomes, yet fishers still engage in competition for the most productive spot. Competitiveness and violence are completely eradicated when *first-to-announce, first-in-right* rules are instituted. Furthermore, this rule configuration promotes fairness in the announcements made by the appointed fisher (i.e. it goes to the spot it said it would go to). This example illustrates an automated analysis of how additional rules can improve outcomes in formally modelled social interactions.



(a) Default rule configuration.

(b) *First-in-time, first-in-right* rule configuration.

(c) *First-to-announce, first-in-right* rule configuration.

Figure 3: EFG semantics for the fishers example.

4 Conclusions

Using the ASL and its complementary game engine, formal descriptions of social interactions can be written, and their outcomes automatically predicted and evaluated. Together, the two constitute a computational model of the IAD framework and allow practitioners to automatically perform *what-if* analysis from a rule configuration to the outcomes most incentivized by it.

This work opens the door to several extensions, like the introduction of *information* rules and dynamic boundary and position rules. Additionally, links between several action situations can be modelled, where the outcomes of an action situation affects the variables that determine the structure of another.

Acknowledgments

This work has been supported by the EU WeNet project (H2020 FET Proactive project #823783), the EU TAILOR project (H2020 #952215), the RHYMAS project (funded by the Spanish government, project #PID2020-113594RB-100), the VALAWAI project (Horizon #101070930) and the VAE project (Grant no. TED2021-131295B-C31 funded by MCIN/AEI /10.13039/501100011033 and by the European Union NextGenerationEU/PRTR).

References

[de Jonge and Zhang, 2021] 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.

[de Jonge *et al.*, 2017] 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.

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

[González-Díaz *et al.*, 2010] Julio González-Díaz, Ignacio García-Jurado, and M. Gloria Fiestras-Janeiro. *An introductory course on mathematical game theory*. American Mathematical Society and Real Sociedad Matemática Española, Providence, Rhode Island, USA and Madrid, 2010.

[Montes *et al.*, 2022] Nieves Montes, Nardine Osman, and Carles Sierra. A computational model of Ostrom's Institutional Analysis and Development framework. *Artificial Intelligence*, 311:103756, 2022.

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

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

[Ostrom, 2011] Elinor Ostrom. Background on the institutional analysis and development framework. *Policy Studies Journal*, 39(1):7–27, 2011.

[Schiffel and Thielscher, 2014] 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.

[Thielscher, 2017] Michael Thielscher. GDL-III: A description language for epistemic general game playing. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 1276–1282. International Joint Conferences on Artificial Intelligence Organization, aug 2017.