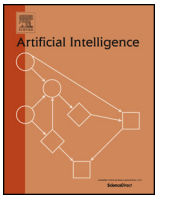
Artificial Intelligence 311 (2022) 103756

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint



A computational model of Ostrom's Institutional Analysis and Development framework

Nieves Montes *, Nardine Osman, Carles Sierra

Artificial Intelligence Research Institute (IIIA-CSIC), UAB Campus, Carrer de Can Planas, Zona 2, 08193 Bellaterra (Barcelona), Spain

ARTICLE INFO

Article history:
Received 29 November 2021
Received in revised form 28 June 2022
Accepted 1 July 2022
Available online 8 July 2022

Keywords:
Institutional Analysis and Development framework
Rules
Normative multiagent systems
Game theory
Logic programming

ABSTRACT

The Institutional Analysis and Development (IAD) framework developed by Elinor Ostrom and colleagues provides great conceptual clarity on the immensely varied topic of social interactions. In this work, we propose a computational model to examine the impact that any of the variables outlined in the IAD framework has on the resulting social interactions. Of particular interest are the rules adopted by a community of agents, as they are the variables most susceptible to change in the short term. To provide systematic descriptions of social interactions, we define the Action Situation Language (ASL) and provide a game engine capable of automatically generating formal game-theoretical models out of ASL descriptions. Then, by incorporating any agent decision-making models, the connection from a rule configuration description to the outcomes encouraged by it is complete. Overall, our model enables any community of agents to perform *what-if* analysis, where they can foresee and examine the impact that a set of regulations will have on the social interaction they are engaging in. Hence, they can decide whether their implementation is desirable.

1. Introduction

The Institutional Analysis and Development (IAD) framework is a conceptual toolbox put forward by Elinor Ostrom and colleagues in an effort to identify and delineate the universal common variables that underlie the immense variety of human interactions [1]. The framework identifies *rules* as one of the core constructs that determine the structure of interactions, and acknowledges their potential to steer a community towards more beneficial and socially desirable outcomes.

This work presents the first attempt to turn the IAD framework into a computational model that allows communities of agents to perform *what-if* analysis on a given rule configuration. To do so, we define the Action Situation Language (ASL) whose syntax is highly tailored to the components of the IAD framework and that is used to write formal descriptions of social interactions. The ASL is complemented by a game engine that generates the semantics of social interactions as extensive-form games (EFGs). These EFGs can then be analyzed with the standard game-theoretical tools to predict which outcomes are being most incentivized, and evaluated according to the overall social benefit they bring about. All the code to go along with this work is open-sourced under an MIT license on the AI4EU platform and GitHub. Beyond the implementation of the fundamental algorithms, we include support for customized visualization of the generated game trees.

* Corresponding author.
E-mail address: nmontes@iiia.csic.es (N. Montes).

https://doi.org/10.1016/j.artint.2022.103756

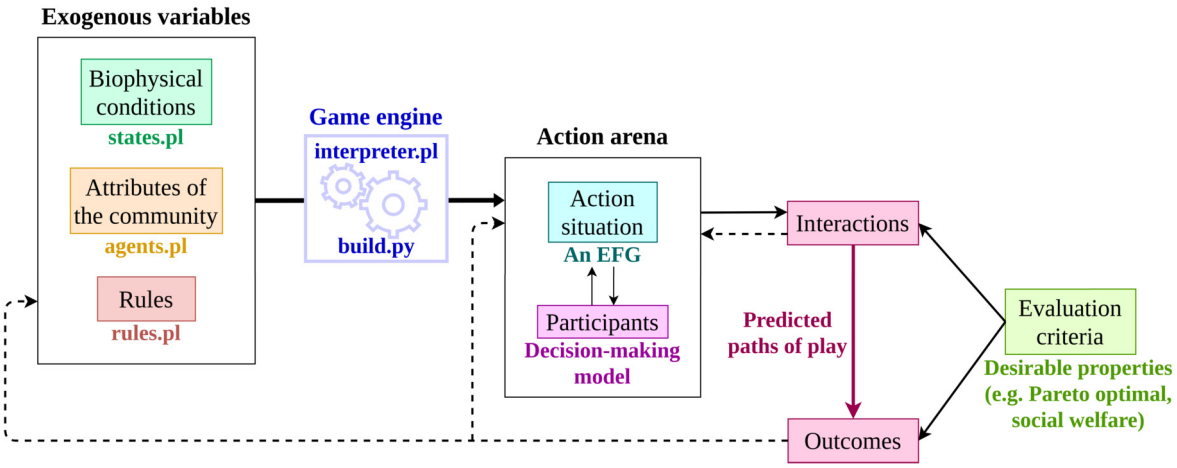


Fig. 1. Outline of the Institutional Analysis and Development framework, adapted from [2, p. 15]. Colored text outside boxes indicates either the scripts that contain information on the boxed component, or the game-theoretical concepts that represent it.

This paper is organized as follows. We start by presenting the necessary background on the IAD framework, outline our contributions and review some related work in the rest of this Introduction. Then, we present the syntax of the Action Situation Language in Section 2. Next, in Section 3, we provide a detailed explanation of the process of rule interpretation – a crucial step to turn action situation descriptions into games – and go through the game semantics generation process. The last technical part, Section 4, reviews some issues related to implementation and the integration of the resulting game representations with game-theoretical tools. Finally, we close with some illustrative examples in Section 6 and make our concluding remarks in Section 7.

1.1. The Institutional Analysis and Development framework

Within the field of policy analysis, the Institutional Analysis and Development (IAD) framework, put forward by Ostrom and colleagues [2], represents a comprehensive theoretical effort to identify and delineate the universal building blocks that make up any social interaction. Its outline is presented in Fig. 1. In the center part, any social interaction is referred to as an *action arena*. In it, a set of *participants* (the agents) find themselves in an *action situation*, which is the social space they may enter, take actions in and jointly bring about outcomes.

According to the IAD framework, action arenas are affected by three sets of exogenous variables that jointly combine to structure it (Fig. 1 left). These are the *biophysical conditions*, the *attributes of the community* and the *rules* of the interaction. The first two are fairly straightforward to define. The biophysical conditions refer to the relevant characteristics of the environment where the interaction takes place, such as land topology and location of resources. The attributes of the community encompass variables intrinsically linked to the participants, such as age, gender, ethnicity and/or belonging to one or several subgroups. Last of all, the meaning of the term *rules* is wide enough to require a detailed clarification that we provide below.

The IAD framework acknowledges the four common uses of the term *rules* in everyday language [3, Ch. 6], according to their scope: instructions, precepts, regulations and principles. First, instructions are understood as a set of steps to effectively achieve some desirable outcome in a given context. Good contemporary examples are Ikea assembly guides. In second place, precepts are somewhat similar to instructions, in the sense that they also directly concern the actions to be taken by an agent. However, their scope is more general. Instead of specifying the particular actions that an agent should perform in a specific situation or context, precepts provide widely applicable principles to help guide decision-making in a range of situations. Good examples are the five precepts from the Buddhist faith [4], which should be regarded by any person adhering to Buddhism regardless of the situation they are confronted with.

In the third place, regulations are, possibly, the most intuitive meaning of rules. They refer to statutes and ordinances that constrain or provide alternative avenues for a course of action. Typically, regulative rules are understood as being passed down from a central authority responsible for their crafting and enforcement. However, small communities can also self-impose regulations on themselves in order to ensure sustainability, fairness, and other desirable goals. For example, there are many cases of small communities of fishers, loggers and crop farmers who craft their own regulations regarding how much fish, wood or water is each member entitled to [5].

Finally, the last of the meanings that rules take are as physical principles. These refer to the laws of nature that inevitably play a part in determining what actions and/or outcomes are physically possible and the effects they have on the environment. If you drop an object, it will fall downwards. Additionally, if it is made of a fragile material such as glass or ceramic, it will most certainly break.

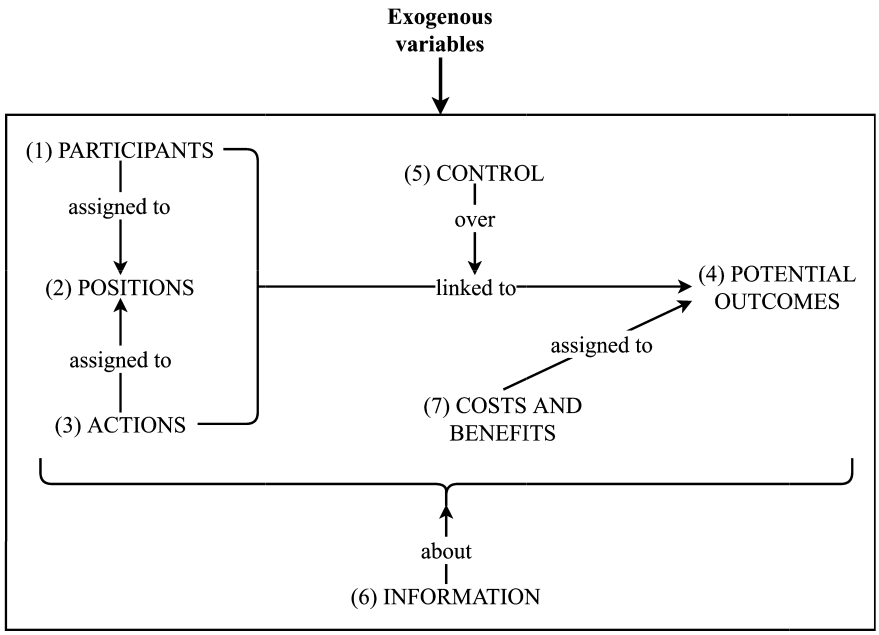


Fig. 2. Internal structure of an action situation, adapted from [2, p. 33].

There is a major difference between the first two and the last two meanings. While instructions and precepts indicate to an agent (either directly as instructions, or indirectly as precepts) what actions to perform provided the situation at hand, regulations and principles condition the structure of the situation itself. Together, regulations and physical principles jointly determine what actions are possible and/or allowed, what their effects are, potential sanctions if a prohibited action is performed (or failure to perform an obliged action) and, consequently, which outcomes may be attained. Once all of that information is gathered, instructions and precepts are invoked on that specific situation to output the particular course of action to take. Hence, instructions and precepts directly target the *actions* to take given a social situation, while regulations and principles determine the social situation itself.

In this work, we take the view that the function of rules is to mold the structure of the situation agents find themselves in, i.e. to modify the incentives and opportunities they face. Hence, the term *rule* will be used to encapsulate both regulations and natural principles. There is a fundamental difference, however, between the two. While regulations are human-made, and hence subject to revision and change, natural principles are not and therefore they are essentially untouchable. We address this distinction in our logical language by distinguishing between *default* rules and additional regulations through a priority relationship between rule statements. Also, we make the distinction between rules that reflect natural principles and the biophysical conditions introduced early on. Physical 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).

In the computational model of the IAD framework we present, we leave out instructions and precepts, since our Action Situation Language does not include an avenue to model them in a systematic manner. We make this choice because we are interested in the constructs that shape the social interactions (i.e. regulations and physical laws). Therefore, this work is not concerned with the individual decision-making (which takes into account instructions and precepts) that agents perform once they are faced some situation. For decision-making, we rely on game-theoretic schemes, which are directly applicable on the extensive-form game representations of social interactions that are automatically built by our tool.

In addition to identifying the variables that condition an action arena, the IAD framework also determines the components that together make up any action situation. There are in total seven variables at play (see Fig. 2): (1) the participants who are allowed to enter; (2) the positions or roles that they take on; (3) the actions assigned to those roles; (4) the potential outcomes that may be reached; (5) the linkage between actions (or sequences of actions) to outcomes and the control that agents have over it; (6) the information available to participants about all other variables (including what information is available to *others*); and (7) the material reward and costs assigned to outcomes and/or actions.

Once participants populate an action situation, it becomes fully instantiated. By introducing some decision-making model for every agent (such as traditional rationality notions like the Nash equilibrium), a prediction of how the interaction is expected to play out and the eventual outcomes (the “Interactions” and “Outcomes” boxes in Fig. 1) that are likely to be reached can be constructed. Finally, these outcomes can be evaluated in terms of some desirable properties (“Evaluation criteria” box in Fig. 1), such as optimality, efficiency, or various metrics of social welfare [6].

If the evaluation is not satisfactory, changes to the exogenous variables should be made in pursue of more desirable outcomes (according to the evaluation criteria of choice). Of the three sets of exogenous variables, biophysical conditions and the attributes of the community are fixed in the short term. In contrast, rules are relatively malleable. In particular, human-crafted regulations are very much susceptible to review and modification. However, the "default" rules (those that reflect natural principles) cannot be changed.

Despite being introduced several decades ago [7], the IAD framework is currently being used in policy analysis studies. Most recent examples include scenarios of pollution and waste management in widely diverse areas of the world [8,9] as well as conservation policy [10]. Research efforts on the theoretical front are also on-going in order to better integrate the IAD framework with existing formal legal systems [11].

1.2. Contributions

The main contribution of this work is a computational model of Ostrom's IAD framework. This model enables communities of agents to formally perform *what-if* analysis of potential new regulatory rules they may be considering to adopt. We provide new tools and integrate them with existing concepts, to compose the complete connection from rule specification to evaluation in terms of the joint outcomes that are encouraged by the regulations in place.

In order to write rule configurations in a systematic manner, we present our novel Action Situation Language (ASL). This is a machine-readable logical language (implemented in Prolog) whose syntax is highly tailored to the exogenous variables outlined in the IAD framework (see Fig. 1). ASL is complemented by a game engine that takes as input a valid action situation description and automatically generates its semantics as an extensive-form game (EFG). EFGs are abstract and very general models, prevalent in the microeconomics field [12, Ch.9], that can be instantiated to represent a wide variety of social interactions among an arbitrary number of agents. Although environmental and community attributes also play a role in generating the EFG semantics, we are particularly interested in the impact that rules have on the resulting formal model. In fact, an essential component of the game engine is a rule interpreter, whose function is to query the rule base, process their implications and solve conflicts between contradicting rules.

In fact, the two main innovations we present (ASL plus its game engine) bridge the gap between the normative multiagent systems (norMAS) and game theory fields. In norMAS, a great deal of work has been devoted to the study of *norms*, *rules* and other constraining mechanisms to achieve coordination and socially beneficial behavior among autonomous agents [13–15]. 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 MAS literature (e.g. [16,17]). However, in game theory, the rules that configure the structure of the interaction become irrelevant once the formal model has been built, and they are often expressed in non-systematic, plain natural language. With ASL, such rules can be expressed in a systematic manner and their semantically equivalent formal game is automatically generated by the game engine.

The choice of EFGs as the semantics for an ASL description is motivated by the availability of many game-theoretical solution concepts, such as traditional rationality notions (e.g. Nash or correlated equilibrium, subgame perfect equilibria, etc.) and social properties of outcomes (e.g. Pareto efficiency, social welfare), that can be readily applied to any model built by the game engine. Due to the prevalence of these well-established concepts in the game theory literature, we do not see the need to provide new solution concepts of our own. Introducing such models of agent decision-making (either "rational" in the traditional sense or not) amounts to modeling the *participant* component of an action arena (see Fig. 1). This step then paves the way to compute the most likely outcomes and evaluating them according to their optimality, efficiency, or social welfare. At this point, the process that takes in a rule configuration and evaluates its impact is complete, and the community of agents involved is informed about the repercussion that such regulations would have on them, were they to be adopted.

1.3. Related work

Originally, the IAD framework was complemented by the Institutional Grammar (IG) [18]. The IG parses institutional statements (which include strategies, norms in the sense of conventions, and regulative rules) into five fields: the attributes (A) of the participants to whom the statement applies; the deontic (D) modality (permitted, forbidden or obliged); the aim (I) of the statement, meaning the action or outcome to whom the deontic applies; the condition (C) under which the statement applies; and the or-else (O) field which states the consequences of non-compliance. Put together, these fields constitute the ADICO syntax. The three types of institutional statements are distinguished by the fields that are necessary to describe them: AIC for strategies, ADIC for conventions and ADICO for rules. Lately, the IG has spurred renewed interest, with extensions to the original proposal including the nesting of statements [19] and the distinction between different levels of granularity in the parsing [20].

Although the early version of the IG did contain examples of formal games built from institutional statements [see 2, Ch. 5-6], no attention has been paid at automating this process, as the ADICO syntax is not designed as a machine-readable language. However, one of its most interesting features, which we will import into ASL to some extent, is the classification of rules based on the component of the action situation that they target, according to the aim (I) field.

Although not being machine-readable, some works have attempted to make the ADICO syntax operational in agent-based models [21,22]. There, the ADICO syntax is used to represent agents' strategies and shared conventions. However, these works are limited in scope, since they use very restricted forms of institutional statements (AIC and ADIC statements obtained from combinations of a pre-defined set of possibilities for every field) and only target the modeling of common-pool resource situations, i.e. natural resources that are jointly exploited by a community of farmers, fishers, loggers, etc., and whose easy access makes it very difficult to forcefully exclude anyone from accessing it [5]. Although the IAD framework indeed accounts for the analysis of this type of scenarios, it is intended to identify and analyze the components of a wide variety of social interactions. Due to the limitations in these previous works, we choose not to build on top of them and move away from the ADICO syntax by defining our own machine-readable language which is able to model a large range of action situations by leveraging the generality of game theoretical models.

Another work with the same objective as ours (turning the IAD framework into a general-purpose operational computational tool) has been developed by [23] as the Modeling Agent systems based on Institutional Analysis (MAIA) framework. Itsworkflow is somewhat similar to ours: input an action situation description into a web application (we feed an ASL description into the game engine) that automatically generates an executable script for an agent-based simulation (our engine generates a game theoretical model). Beyond technical differences ([23] employs Java plus HTML, while we use a combination of Prolog and Python), our contributions diverge in the encoding of institutional statements, as [23] stick to the ADICO syntax, while we propose a new *if-then-where* syntax.

However, the most significant difference between [23] and the present work lies in the approach to the "Participants" component in Fig. 1. Our computational model of the IAD framework is agnostic with respect to the decision-making model participants follow once they find themselves within an action arena. Hence, modeling participants and generating an action situation representation are independent tasks and, in principle, the same decision-making model can be applied across a wide diversity of situations, e.g. Nash equilibria computation can be applied to any extensive-form game. In contrast, the MAIA framework requires a criterion for decision-making to be explicitly provided as an input to their simulation generator, and therefore it needs for the participants to be modeled beforehand and crafted for every particular simulation. Such criterion is tailored to the context at hand, and is not, in principle, exportable to other situations.

On another front, the field of General Game Playing (GGP) within the AI community has come up through the years with machine-processable languages for the specification of general games. Most prominently, the Game Description Language (GDL) [24] is a high-level language for the specification of games with a finite number of players and legal moves. GDL provides compact descriptions of deterministic classical games (such as chess and checkers) and also admits a form of restricted imperfect information in the form of simultaneous moves, a feature that we incorporate into our language.

Since its creation, some extensions have been added to GDL in order to improve its expressive power. Most notably, GDL-II [25] incorporates the possibility of imperfect information and random moves by nature, although limiting those to a uniform probability distribution. Later, yet another addition resulted in the introduction of GDL-III, where epistemic games in which the rules depend on the knowledge of the agents can be represented by introducing player introspection [26]. Beyond game playing, GDL (in its original version) has been used for more socially relevant applications, such as mediated dispute resolution [27] and automated negotiation [28].

For comparison purposes, ASL and GSL descriptions of the benchmark Iterated Prisoner's Dilemma game are displayed in Listings 1 to 4. Although both GDL and our ASL are logical languages for game specification, some of the features of ASL make it much better suited than GDL for modeling socioeconomic interactions. First, when using the term "rules of the game" in relation to GDL, it is referring to the complete game description (i.e. the logical program). In contrast, by "rules" in this work we refer to one of the components describing an action situation, i.e. to the exogenous variable "rules" in Fig. 1, separate from biophysical conditions and attributes of the community.

Second, although the authors in [25] include a qualitative description of a procedure to turn GDL descriptions into extensive-form games (and vice-versa), this is not a central contribution of their work. Instead, they provide a logic for reasoning about GDL game descriptions based on a variant of the Situation Calculus [29]. Differently, we put a lot of focus on the interpretation and the translation of ASL descriptions into EFGs. These are the most prevalent models to represent social interactions in microeconomics and policy analysis (see the examples in Section 6). They are abstract and general enough to capture a wide variety of interactions, while at the same time being amenable for analysis by implementing notions of rationality that are prevalent across the social sciences.

Finally, the feature that sets ASL apart from GDL is the fact that ASL descriptions are meant to be extensible. That is, an ASL description is intended to be expanded with additional higher-priority rules. In other words, the same ASL description can give rise to two different multiagent interactions, depending on whether new rules in addition to the default ones are included or not. Differently, GDL game descriptions are static and not meant for modification. This is reflected in the fact that GDL does not incorporate any mechanism to solve conflicts between rules, while ASL does.

A game description system previous to GDL was the Game Language (Gala) [30]. The focus of the Gala system was on the efficient computation of solutions of large imperfect information game trees, and it suffers from some of the same drawbacks that make it unsuitable for the representation of socioeconomic interactions. In particular, Gala does not consider the rules of the game separate from other relevant exogenous variables either, nor are its descriptions meant to be extended.

2. ASL syntax

Our intention is to define the syntax of ASL as fully machine-readable, yet also relatively syntactically friendly to make it accessible to social science scholars. In order to completely describe an action situation, our language must specify the three sets of exogenous variables that affect it (see Fig. 1):

- **Attributes of the community**: the agents susceptible of taking part in the interaction, plus any relevant characteristics: age, gender, ethnicity, etc.
- **Biophysical and environmental conditions**: land topology, location of resources, etc.
- The **rules** structuring the situation, in particular the following four *types*, according to which aspect of the action situation they address:
  - **Boundary rules**: which agents are allowed to enter the action situation. For example, in many countries it is required to be over 18 years old to participate in an electoral process.
  - **Position rules**: what roles do the participants take on. For example, candidate, voter, etc.
  - **Choice rules**: what actions are available to the various roles under the current conditions. For example, an agent with the role *voter* can take the action to vote for one (or none) of the candidates.
  - **Control rules**: what are the effects of those actions. In a majority rule electoral process, the candidate with the most votes gets appointed to the position in contention.

Additionally, the following information is also necessary:

- The initial conditions when the interaction starts.
- The termination conditions under which the interaction halts.
- Which facts describing the state of the system can be simultaneously true (for example, an agent cannot be at two different locations at the same time).

As explained in Section 1.1, we consider rules in the sense of regulations and physical principles. Concerning the former, rule statements in ASL completely encapsulate human-made regulations. In fact, boundary, position and choice rules (which deal with providing agents with access to the social interaction, a role in it and actions to affect it, respectively) are not in any way related to the natural principles governing the environment. Concerning physical principles, these are captured both by control rules that dictate the dynamics of the system (see Section 3) and incompatibilities between facts, which are expressed through a dedicated predicate symbol.

ASL descriptions follow the standard syntax of logic programming and are thus composed of constant symbols, function symbols, predicate symbols and variables. Expressions are classified as one of the following:

- A *term* is a variable, or a function symbol with terms as arguments.
- A *literal* is a predicate symbol (or its negation) with terms as arguments. Terms and literals that do not contain any free variables are called *ground terms* and *ground literals* respectively.
- A *clause* is an expression of the form $h :- b_1, ..., b_n$, where the head h is a non-negated literal and the body $b_1, ..., b_n$ are literals, with the meaning that $b_1, ..., b_n$ together imply h.

Also, it is worth mentioning *lists*, which are ordered sets of elements are enclosed by "[" and "]" (the empty list is written as "[]"). Also, the anonymous variable is represented by a single underscore `_`. Its different occurrences may represent different literals. Regular variables, in contrast, start with a capital letter (e.g. `Agent, Action`) and their occurrences are all instantiated to the same ground literal within the scope of a clause.

ASL descriptions define, primarily, how a multiagent system evolves and transitions between states. A state s_t is defined as a finite set of ground literals, $s_t = \{f_1, ..., f_n\}$ (if p is an m-ary predicate symbol and $\alpha_1, ..., \alpha_m$ are ground terms, then $p(\alpha_1, ..., \alpha_m)$ is a ground term). The predicate symbols used to describe a particular action situation depend on the domain at hand and are a design choice by the user. Hence, the truth of a fact (i.e. a ground literal) f_i in state s_t holds iff $f_i \in s_t$. How the facts are initialized and evolve is a matter for the building of the EFG semantics from the ASL description (Section 4.2) and the interpretation of control rules (Section 3.2).

The keywords of ASL are gathered in Table 1. Most of these appear in `rule/4` arguments, and only `agent/1`, `initially/1`, `terminal/0` and `incompatible/2` are used as standalone predicates. In fact, the `agent/1` predicate symbol appears both within `rule` statement and as a standalone predicate. We start by reviewing the predicates that do not appear within rules. First, `agent(Ag)` denotes `Ag` as an individual susceptible of entering the action situation.

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

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

Thus, this predicate provides information on the attributes of the community. If needed, domain-dependent predicates of the type `feature_name(Ag,Val)` can be added to encode agent attributes. For example, `age(alice,34)`.

Second, `initially(F)` indicates that literal `F` holds true at the start of the interaction, prior to any action being executed. For example, to indicate that at the start of the interaction, all agents, regardless of their role, are at the origin of coordinates, we need to include the clause `initially(at(Ag, position(0,0))) :- role(Ag,_).` `terminal/0` plays the opposite role, as it returns true whenever the conditions for halting the interaction are met. For example, to indicate that the interaction stops the moment an agent makes it to a finish line placed horizontally at some height y_{fl}, we need to include the clause `terminal :- at(Ag, position(_,Y)), Y>= y_fl.`

Finally, `incompatible(F,L)` states that literal `F` cannot be simultaneously true with the literals in list `L`. Formally, `incompatible`(f, L) means that $f \notin s_t$, where $s_t = \{l_i \mid l_i \in L\}$ is the state built from the literals in list L. For example, to indicate that agents cannot be at two different positions simultaneously, we need to include the clause `incompatible(at(Ag,Pos1),L) :- member(at(Ag,Pos2),L), Pos1\==Pos2.`

This example may raise the doubt of why we have chosen to have the second argument to `incompatible/2` literals be a list, instead of just a literal. We believe that having a list allows for greater flexibility in ASL descriptions. For example, suppose fact f_1 is only incompatible with facts f_2 and f_3 *simultaneously*, meaning that f_1 cannot be part of a state only if f_2 and f_3 are both part of it. This statement could not be expressed if the second argument to `incompatible/2` were a single literal. With our current syntax, it can be captured by the clause `incompatible(`f_1`,L) :- member(`f_2`,L), member(`f_3`,L).`

We move on now to `rule/4` predicates. All of its clauses, regardless of the component they target, follow the general template in Fig. 3, with the following four arguments:

1. An identifier `Id` that denotes the action situation where the rule is to be applied.
2. The `Type` of the rule, one of either *boundary*, *position*, *choice* or *control*.
3. The `Priority` of the rule. This is a non-negative integer that determines which statement is to prevail in case several rules lead to contradicting consequences. The rule statements that are supposed to reflect the physical principles of the domain are assigned priority 0 and are referred to as the *default rules*, while additional human-made regulations have strictly positive priorities. The reserved overwriting operator ~ is introduced in order to have high priority rule nullify the effects of lower priority rules. We use the term *overwriting* instead of *negation* operator since ASL, as a logic programming language, follows negation as failure.
4. The content of the rule is expressed with an `if-then-where` statement (the three are all ASL reserved operators, see Table 1). The content of the `Condition` and `Consequence` fields is subject to syntactic constraints according to the type of the rule in question. We review these syntactic constraints in detail in the next section. The `Constraints` field always consists of a list of literals and constraints, 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.

Besides predicate symbols, Table 1 also displays reserved operators, all of which appear within the scope of `rule/4` literals. Since ASL is implemented in Prolog, action situation descriptions can also make use of built-in Prolog predicates (notably `member(Elem,List)`, which has already been invoked) and operators, such as those for comparing terms. The least familiar of these are `Term1@<Term2` (also `@<=`, `@>`, `@>=`), which is interpreted as `Term1` preceding `Term2` in the standard order of terms (i.e. also considering characters). Additionally, The Prolog library for constraint logic programming

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

Fig. 3. General syntax for *if-then-where* rules.

Table 2
Syntactic restrictions for the `Condition` and `Consequence` fields for every of the proposed rule types. α stands for a literal, 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`]		

over real numbers is autoloaded with the ASL interpreter, part of the game engine. This library provides support for numerical constraints with syntax {*Constraints*} (for example, {`Payoff` < 10}).

2.1. Syntax by rule type

As introduced, ASL considers four rule types (boundary, position, choice and control) that target different action situation components in Fig. 2. First, the boundary rules are aimed at regulating the *participants* (1) components of action situations, as they designate which agents are able to enter the interaction. Second, position rules are responsible for assigning participants to their *roles* or *positions* (2). A participant may take on multiple roles. Third, choice rules assign *actions* (3) to roles, not to participants nor agents directly. Hence, an agent that is designated as a participant but is not assigned any role is irrelevant to the evolution of the interaction. Finally, control rules state what is the effect of actions on the system. Hence, this last rule type is directly responsible for the *control* (5) component of action situations.

Note that there is some disconnection between our four rule types and the seven variables within an action situation in Fig. 2. There are no dedicated rule types for the outcomes (4), costs and benefits (7), and information (6) variables. For the first two (outcomes, and costs and benefits), we argue that control rules are in charge. As they effectively regulate how does the state of the world evolve, they are also indirectly determining what *outcomes* are possible. Additionally, if one considers monetary and material rewards to be relevant in the current action situation, it is just enough to introduce a *payoff* predicate, initialized, for example, with `initially(payoff(Ag,0)) :- role(Ag,some_role)`. Then, its evolution can be regulated with control rules, that map (possibly joint) actions to monetary gains. A simple example of the use of control rules to regulate payoffs comes with the Iterated Prisoner's Dilemma example in Section 2.3.

As for the information component, it is left unaddressed in this early version of ASL. As we explain in Section 4, we only consider a restricted version of imperfect information in the semantics of any ASL description (much like in GDL game specifications). Although introducing information constraints to limit players' observability would certainly make for an interesting extension, we do not include it here as we anticipate that it would greatly increase the complexity of the resulting formal games, potentially opening the door for incomplete information and imperfect recall games.

As previously announced, the content of rule statements follows the syntax `if Condition then Consequence where Constraints`, with additional restrictions on the `Condition` and `Consequence` fields depending on the rule type. These restrictions are displayed in Table 2. Note that the boundary, position and choice rules all have an analogous syntax: one `agent/1`, `participates/1` or `role/2` literal as the `Condition`, and `participates/1`, `role/2` or `can/2` as the `Consequence`, respectively. Also, their `Consequence` literal might be preceded by the overwriting operator ∼, although it only makes sense to use it with non-default rules. The overwriting operator ∼ should not be confused with strong negation. The operator ∼ is used to have higher priority rules overwrite lower priority rules in case conflict arises between the consequences of different rule statements. Our computational model still uses the closed-world assumption and consequently negation as failure.

In contrast to the other rule types, control rules may have in their `Condition` multiple `does/2` literals concatenated by the `and` operator to account for the possibility that some effects are only brought about by joint actions, i.e. by having several agents perform some action simultaneously. If one wished to express the fact that several different actions `Act`$_1$ ... `Act`$_n$, performed by several different agents `Ag`$_1$ ... `Ag`$_n$, lead to the same effect (analogous to an `or` operator), one needs to include several control rules, one per each agent-action pair: `if does(Ag1,Act1) then Conseq`, ..., `if does(Agn,Actn) then Conseq`.

The `Consequence` of control rules, instead of a single literal, is a list where each of its members consists of (possibly several) literals concatenated with the `and` operator. In contrast with the other rule types, the literals that can be included in the `Consequence` field of control rule are not syntactically restricted. In general, they contain domain-specific predicate symbols, chosen to specifically reflect the state properties that are relevant to the action situation at hand.

The whole conjunction of predicates that makes up one potential consequence is assigned some probability with the operator `withProb`. The introduction of this operator is completely motivated by the desire to make control rules expressive enough as to encapsulate non-deterministic environments, where actions performed by the agents may have random

effects. In order for a control rule to be valid, the probability distribution over the potential consequences must be well-defined, i.e. p_i must fall in the range $[0, 1]$ for all i and must add up to unity.[1] Of course, deterministic environments can be expressed by having all control rules have one single consequence in their `Consequence` list, assigned probability equal to one.

In addition to stochastic effects, actions performed by agents may have different effects depending on the *context* where they are performed, *i.e.* the facts that hold true in the current state of the system. Such dependencies are handled through the `Constraints` field. Suppose the joint action profile $\mu = \texttt{does}(ag_1, ac_1)\ \texttt{and}\ \texttt{does}(ag_2, ac_2), \ldots$ has consequences c_1 when the state of the system is in s_1, consequences c_2 when in s_2, etc. To reflect this, the action situation description would need to include one control rule for every c_i, s_i pair, with the following content: `if` μ `then` c_i `where` $[s_i]$.

Regimentation and deontic modalities In the norMAS field, the term *norms* often refers to constraints and prescriptions on the behavior of agents intended to coordinate their actions [13]. Typically, norms are represented using a *deontic modality* (prohibited (P) or obliged (O), following [31]). Although the terminology "deontic modality" is inspired by deontic logic [32], P and O are rather used as operators to express constraints on actions (as in [33]) or states (as in [34,35]). This is in line with the operational semantics that we define for ASL in Section 4. Additionally, such normative constraints can be regimented or not [36]. In regimented domains, the nature of the application allows for the perfect enforcement of the desired constraints. Meanwhile, in non-regimented domains, some monitoring and sanctioning mechanism is implemented in order to deter agents from violating the norms.

In this short section, we provide guidelines on how prohibitions and obligations on *actions* can be specified through ASL rule statements, both in the regimented and sanctioning versions. We should note that, in order for any action to be possible in the first place, it needs to be explicitly included in a choice rule and assigned to the role capable of executing it.

Regimented constraints are addressed through choice rules, as they effectively allow or deny the possibility to execute some action. For example, suppose that the default rules (that model the "unconstrained" situation) are to be overwritten in order to prohibit some action, by banning the possibility of executing it. This situation is equivalent to introducing a higher priority choice rule of the form:

```
rule(Id,choice,N,if role(Ag,Role) then ~can(Ag,forbidden_action) where [ConditionsForProhibition
                                                                        ]).
```

where `N` is a strictly positive integer and `Role` is the position (following Fig. 2) that was originally assigned the action in question.

Similarly, regimented obligation can also be modeled through choice rules. To do so, we use the following equivalence between prohibition and obligation:

$$O\left(a_i \mid \psi\right) \Leftrightarrow P\left(a_j \mid \psi\right),\ \forall a_j \neq a_i \tag{1}$$

which states that, provided that some condition ψ holds true at the current state, the obligation to perform action a_i is equivalent to the prohibition of performing any other action a_j. Then, a regimented obligation can be expressed in ASL as:

```
rule(Id,choice,N,if role(Ag,Role) then ~can(Ag,Ac)
where [Ac\=obliged_action,ConditionsForObligation]).
```

Non-regimented norms assume that it is not possible to completely ban the execution of forbidden actions, and instead they attempt to discourage agents away from them. In ASL, this approach can be expressed through control rules. A deterrence for a forbidden action follows the template:

```
rule(Id,control,N,if does(Ag,forbidden_action) then [Punishment withProb P,
       NoPunishment withProb 1-P] where [ConditionsForProhibition]).
```

where `P` is the probability of detection violation.

Similarly to regimented norms, the leap from non-regimented prohibition to obligations is made thanks to Equation (1):

```
rule(Id,control,N,if does(Ag,Ac) then [Punishment withProb P, NoPunishment withProb 1-P]
                  where [Ac\=obliged_action,ConditionsForObligation]).
```

The addition of rules following the templates provided in this section is useful if one wishes to analyze the effects of introducing prohibition and obligation norms, either regimented or not, on a (possibly) unregulated social interaction. That is the reason why the priority `N` must be strictly positive, so it overwrites any conflicting default rules representing the unregulated interaction. This provides a big point of contact between this work and the concerns of the norMAS community [15], that deals with the representation and implementation of norms (obligations and prohibitions) in multiagent systems.

[1] If that is not the case, the game engine that interprets the rules will raise an error, and the game that corresponds to that ASL description will not be generated.

2.2. Valid ASL descriptions

Following the syntax of ASL, a valid action situation description $\mathbb{A}$ can be written as logic program composed of a finite set of clauses. We organize them into three exclusive subsets, one for every set of exogenous variables determining the structure of an action situation (see Fig. 1):

$$\mathbb{A} = \Delta \cup \Sigma \cup \Omega \tag{2}$$

where:

- Δ is the *agents base*, which includes the information on the agents and their attributes.
- Σ is the *states base*, which includes the information on biophysical features, plus clauses with `initially/1`, `terminal/0` and `incompatible/2` predicate symbols in their heads.
- Ω is the *rules base*, which contains the `rule/4` literals (i.e. bodyless clauses).

In order to ensure the unambiguous interpretation of an action situation description, some limitations are placed on the use of the reserved predicates within the body of clauses. Additionally, some minor directives are intended to ensure the clear separation between the community attributes, the biophysical features and the rules into three separate knowledge bases.

Definition 1 *(valid ASL description).* A *valid ASL description* $\mathbb{A}$ is a finite set of clauses split into three exclusive subsets $\Delta \cup \Sigma \cup \Omega$, where:

- `agent/1` predicate symbols appear only as the head of clauses in Δ.
- `initially/1`, `terminal/0` and `incompatible/2` predicate symbols appear only in the head of clauses in Σ. Also, `initially/1` clauses do not contain `can/1` nor `does/2` predicates in their bodies. This restriction reflects the fact that the initial state has to be completely determined *before* agents inspect what actions are at their disposal (`can/2`) or they choose one of those actions to perform (`does/2`). Furthermore, `incompatible/2` does not have a reserved predicate symbol as its first argument.
- `rule/4` predicate symbols appear only as literals (clauses with no body) in Ω, plus they follow the syntactic restrictions exposed in Table 2.

2.3. First example: iterated Prisoner's Dilemma

The best way to get a grasp on the ASL syntax is to go through a simple illustrative example. In this section, we show how the iterated version of the benchmark Prisoner's Dilemma game can be specified in ASL. Additionally, we take the opportunity to compare ASL to GDL in more detail, by differentiating the descriptions that both languages make of this benchmark. Its simplicity and familiarity to a wide range of audiences makes it a perfect candidate to be the first described with our language. Other more sophisticated examples are presented in Section 6.

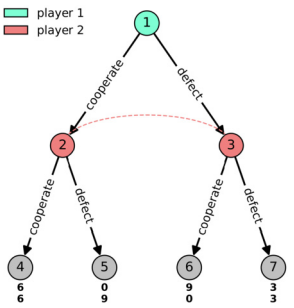
In the Prisoner's Dilemma game, two agents are put in identical positions, where they can choose one of two actions (either "cooperate" or "defect"), with no prior opportunity to communicate with one another. The normal and extensive-form representation of the game appear in Fig. 4. If an agent cheats on the other, the defector receives a large temptation payoff ($T = 9$) at the expense of the sucker ($S = 0$). If the two agents cooperate, they both receive identical reward payoffs ($R = 6$) that are larger than the punishment payoff ($P = 3$) they get when both choose to defect. This game has attracted a lot of interest from scholars in a wide range of disciplines because, although the most socially beneficial thing to do is for both agents to cooperate with one another, rational behavior stipulates that they should both defect, leading to worse individual and group reward. In game theoretical terms, the Nash equilibrium is not Pareto optimal. In its iterated version, agents typically start with wealth equal to zero. They play several consecutive rounds of the one-shot game, and their wealth is increased by the reward they get at every round. The interaction is halted after some pre-defined number of rounds have been completed.

The ASL description of the iterated Prisoner's Dilemma game appears in Listings 1 to 3. First, the agents and states knowledge bases appear in Listings 1 and 2 respectively. In the agents base, two agents are declared with no additional attributes. In the states base, the initial conditions are set to zero wealth for all prisoners. Also, a counter for the number of rounds is initialized. Finally, the `incompatible/2` clauses indicate that there can only be one `rounds/1` literal, as well as one `payoff/2` literal per agent in a state.

Second, the rule base is displayed in Listing 3. There are 8 rules in total: 1 boundary, 1 position, 2 choice and 4 control. They are all identified by the tag `ipd` and have priority equal to zero since they are the defaults. The boundary, position and choice rules are all very generic. All agents are allowed to participate by the boundary rules, they take on one role (that of a prisoner) according to the position rules and they are allowed to cooperate or defect with no further constraints by the choice rules.

	cooperate	defect
cooperate	6,6	0,9
defect	9,0	3,3

(a)



(b)

Fig. 4. The Prisoner's Dilemma game in the (a) normal-form and (b) extensive-form representations. Note the use of an information set (defined in Section 4.1) for player 2, denoted with the dashed line joining nodes 2 and 3, to capture the simultaneous nature of the moves.

```
1 agent(alice).
2 agent(bob).
```

Listing 1: Agents base Δ for the iterated Prisoner's Dilemma ASL description.

```
1 initially(payoff(P,0)) :- role(P,prisoner).
2 initially(rounds(0)).
3 terminal :- rounds(N),N>=3.
4 incompatible(rounds(_),L) :- member(rounds(_),L).
5 incompatible(payoff(P,_),L) :- member(payoff(P,_),L).
```

Listing 2: States base Σ for the iterated Prisoner's Dilemma ASL description, played for a total of three rounds.

```
1 rule(ipd,boundary,0,if agent(A) then participates(A) where []).
2
3 rule(ipd,position,0,if participates(A) then role(A,prisoner) where []).
4
5 rule(ipd,choice,0,if role(P,prisoner) then can(P,cooperate) where []).
6 rule(ipd,choice,0,if role(P,prisoner) then can(P,defect) where []).
7
8 % P1@<P2 avoids equivalent instantiations of some control rules
9 rule(ipd,control,0,if does(P1,_) and does(P2,_) then [rounds(M) withProb 1] where [P1@<P2,rounds(
      N),{M=N+1}]).
10 rule(ipd,control,0,if does(P1,cooperate) and does(P2,cooperate)then [payoff(P1,Y1) and payoff(P2,
      Y2) withProb 1] where [P1@<P2,payoff(P1,X1),payoff(P2,X2),{Y1=X1+6,Y2=X2+6}]).
11 rule(ipd,control,0,if does(P1,defect) and does(P2,defect) then [payoff(P1,Y1) and payoff(P2,Y2)
      withProb 1] where [P1@<P2,payoff(P1,X1),payoff(P2,X2),{Y1=X1+3,Y2=X2+3}]).
12 rule(ipd,control,0,if does(P1,cooperate) and does(P2,defect) then [payoff(P1,Y1) and payoff(P2,Y2
      ) withProb 1] where [payoff(P1,X1),payoff(P2,X2),{Y1=X1+0,Y2=X2+9}]).
```

Listing 3: Rule base for the iterated Prisoner's Dilemma ASL description.

The control rules exemplify the use of the `Constraints` field. The first one declares that, in order for the rounds counter to advance, two distinct participants must take some action. The other three control rules have identical structures as they all control the rewards received as a function of the joint actions at every stage of the game. Although there are four possibilities in the Prisoner's Dilemma game (see Fig. 4a), due to symmetry we need only three control rules. The dynamics of this environment are deterministic, hence all control rules have one single consequence with probability equal to unity.

Now that the syntax of ASL has been thoroughly explained, we are able to provide a comparison with that of GDL, whose description for the iterated Prisoner's Dilemma appears in Listing 4. The first small difference is that the GDL description is not split into components like the ASL one. Also, ASL separates the notions of *agent*, *participants* and *role*. In contrast, GDL merges the three concepts and declares any player participating in the game using the predicate `role/1`. Concerning

```
role(alice).
role(bob).

init(payoff(Ag,0)) :- role(Ag).
init(rounds(0)).
terminal :- rounds(N),N>=3.

legal(Ag,cooperate) :- role(Ag).
legal(Ag,defect) :- role(Ag).

next(rounds(M)) :- true(rounds(N)), M=N+1.
next(payoff(Ag1, Y)) :- does(Ag1,cooperate),does(Ag2,cooperate),role(Ag1),role(Ag2),Ag1@<Ag2,true
    (payoff(Ag1,X)),Y=X+6.
next(payoff(Ag1, Y)) :- does(Ag1,defect),does(Ag2,defect),role(Ag1),role(Ag2),Ag1@<Ag2,true(
    payoff(Ag1,X)),Y=X+3.
next(payoff(Ag1, Y)) :- does(Ag1,cooperate),does(Ag2,defect),role(Ag1),role(Ag2),true(payoff(Ag1,
    X)),Y=X+0.
next(payoff(Ag1, Y)) :- does(Ag1,defect),does(Ag2,cooperate),role(Ag1),role(Ag2),true(payoff(Ag1,
    X)),Y=X+9.
```

Listing 4: GDL description of the Iterated Prisoner's Dilemma game in infix syntax.

similarities, the `initially/1` and `terminal/0` clauses in ASL have the same function (and also practically the same spelling) as `init/1` and `terminal/0` clauses in GDL.

Regarding the `rule/4` literals in GDL, boundary and position rules do not have an equivalent in GDL since, as we have already mentioned, this latter language does not distinguish between agents, participants and roles, as it is not grounded in any social science theory but rather developed by and for the General Game Playing community. Choice rules in ASL are analogous to `legal/2` clauses in GDL, as both have the function of assigning actions to roles. Finally, control rules in ASL are analogous to `next/1` clauses in GDL, which contain as argument a literal that will hold true provided that actions in `does/2` predicates are performed and the facts included in `true/1` predicates are true in the current state.

The reader will have noticed that GDL clauses do not contain anything analogous to the rule priority introduced in ASL. This feature, we believe, sets ASL apart from GDL. It allows ASL description to be extended with, for instance, additional norms such as the ones suggested in the previous discussion on obligations and prohibitions. Therefore, it makes ASL much more suitable to the analysis of socioeconomic scenarios, where the interest is on the effect that changes in regulations will have on a given scenario, rather than on the construction of a new interaction model from scratch.

3. Rule interpretation

As introduced early on, an ASL description has its semantics automatically generated as a formal game model by a computational engine. To do so, the game engine has to repeatedly interpret the rules in place. This task is performed by the "interpreter.pl" script (see Fig. 1).

In this section, we go through the rule interpretation process in detail. We split it into two steps: (1) rule activation and (2) processing of consequences. Throughout, we denote an action situation description, split into its three components, again as $\mathbb{A} = \Delta \cup \Sigma \cup \Omega$. On several occasions, this set of clauses is expanded by ground literals on the participants $\phi = \{\texttt{participates}(ag_1), ...\}$, their roles $\rho = \{\texttt{role}(ag_1, r_1), ...\}$, the current state of the system $s_t = \{f_1, ..., f_n\}$ (where n is the number of necessary literals to completely describe the state of the system) and the actions executed by agents $\mu = \{\texttt{does}(ag_1, ac_1), ...\}$. All of these sets ($\phi$, ρ, s_t and μ) have a finite number of elements and do not belong to any of the components of the description (Δ, Σ or Ω), but are appended to the action situation description as a whole at concrete moments during the game construction procedure. As we will see in Section 4, the participants ϕ and roles ρ remain constant throughout the interaction, while the joint actions μ and the current state s_t change as the system evolves. Since the literals in s_t and μ change as the system evolve, we refer to them as *fluents*.

First, the interpreter has to find which rules in Ω are *active* given the current state of the system. In general, `rule/4` statements contain free variables that have to be instantiated to constants given the ASL description $\mathbb{A}$ and the current state of the system. When processing *boundary* rules, $\mathbb{A}$ does not need to be expanded. However, when processing *position* rules, the set of participant atoms ϕ has to be appended to $\mathbb{A}$. For processing *choice* rules, in addition to the set of participants ϕ, the set of roles ρ and the current state of the system s_t are necessary. Finally, for *control* rules, all of the previous extensions are necessary, plus the fluents denoting what actions μ agents are taking. When a rule is fully instantiated, we say that it has been *activated* or *triggered*. A rule may be activated multiple times, as many as possible instantiations of its free variables are possible.

The clause responsible for finding out active rules is `query_rule(Rule)`, shown in Listing 5. Its examination reveals that, in fact, a rule in ASL:

```
1 query_rule(rule(ID,Type,Priority,if Condition then Consequence where Constraints)) :-
2         rule(ID,Type,Priority,if Condition then Consequence where Constraints),
3         maplist(query,[Condition|Constraints]).
4
5 query(Q) :- call(Q).
6 query(A and B) :- query(A),query(B).
```

Listing 5: Prolog interpreter predicates to find which rules are activated given the current state of the system s_t.

```
rule(..., if Condition then Consequence where [Constraint1, Constraint2 ...]).
```

is equivalent to a traditional clause:

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

However, beyond its friendlier syntax as compared to traditional clauses, `rule/4` statements contain a `Type` argument, that allows to activate different kinds of rules depending on the step of the game construction process that the engine is in. Additionally, the `Priority` field helps sort out conflicts between norms with contradicting or incompatible consequences, and also filter out rules whose priority is over some threshold.

The rule activation step is common to all rule types, and it helps retrieve the consequences that rules have and that will have some effect on the resulting interaction. But first, these consequences need to be processed. Since boundary, position and choice rules all have analogous syntactic restrictions, the processing of their derived consequences is also shared. We deal with these three separately from the more complex control rules.

3.1. Processing of boundary, position and choice rules

As displayed in Table 2, boundary, position and choice rules have similar syntactic restrictions as their `Consequence` field contains a single literal. The consequences for all of these rule types are processed by the function Get-Simple-Conseqs (see Algorithm 1 in Appendix A). First, the (extended) action situation description is queried in order to find the active rules of the input type (Line 3). The $pr : f$ pairs are stored (where pr is the integer rule priority and f is the ground literal derived from the rule's `Consequence` field), and those whose priority is over some input threshold are ignored (Line 4).

The inclusion of such a *threshold* argument is, essentially, for convenience purposes. The reader will note that such an argument appears in all other algorithms presented in this work. The aim of this threshold is to allow an ASL user to write a single action situation description with rules of several priorities, and effortlessly obtain the formal representations of the social interaction when different rules are included, simply by tuning the *threshold* argument. This avoids the need to write (or uncomment) higher priority rules every time their impact in the interaction of interest wants to be assessed.

Next, the $pr : f$ pairs are ordered in descending order of `Priority` value (ties broken arbitrarily, Line 5). Then, the atom f derived from `Conseqs` is added to the output set if neither that same predicate nor its overwriting (with the prefix operator $\sim f$) have already been added to the output set by a higher priority rule (Lines 7-8). Finally, literals preceded by the overwriting operator are deleted before the set of facts is returned (Lines 9-10).

The rule interpreter relies on the fact that boundary, position and choice rules are *sound*. This means that Ω cannot contain two rules of the same type and priority with contradicting consequences (i.e. one rule overwrites the consequences of the other). For example, the following two rules should not both be included in the database:

```
rule(...,boundary,1,if agent(Ag) then participates(Ag) where [age(Ag,N),N>18]).
rule(...,boundary,1,if agent(Ag) then ~participates(Ag) where [age(Ag,N),N>18]).
```

If such rules were included, the output set of Get-Simple-Conseqs(..., boundary, 1) would not be deterministic, as it would depend on how the ties during the sorting procedure are broken.

Note that the ASL description $\mathbb{A}$ alone, without any additional fluents, is enough to process only the boundary rules. The set of participants $\phi = \{\texttt{participates}(ag_1), ...\}$ should be added to the action situation description before processing the consequences of position rules. Likewise, in order to process the consequences of choice rules both the roles $\rho = \{\texttt{role}(ag_1, r_1), ...\}$ and the current state $s_t = \{f_1, ..., f_n\}$ have to be added to $\mathbb{A}$. This data is necessary to ensure the proper activation of the various rule types.

To conclude the interpretation of simple rules, we illustrate how Algorithm 1 enforces the constraints of regimented prohibitions and obligations presented in Section 2.1, which are expressed through choice rules. We exemplify regimented prohibitions, the procedure for regimented obligations is analogous. Consider the following two choice rules:

```
rule(...,choice,0,if role(Ag,role) then can(Ag,ac) where []).
```

```
rule(...,choice,1,if role(Ag,role) then ~can(Ag,ac) where [condition_for_prohibition]).
```

The first is a default rule that assigns a generic action to a role regardless of the circumstances (note the empty list in the `Constraints` field). The second is a higher priority rule that bans these actions under some conditions.

Consider the execution of Algorithm 1 (with argument *type* = *control*) with an action situation description including the two rules above. Additionally, assume that the state description $s_t = \{f_1, ..., f_n\}$ fulfills the conditions for the second rule to be activated (i.e. $f_1 \wedge ... \wedge f_n \models \texttt{condition\_for\_prohibition}$). Then, kv on Line 6 of Algorithm 1 is assigned to:

$$kv \leftarrow [1 :\sim \texttt{can}(ag, ac), 0 : \texttt{can}(ag, ac)]$$

where we are assuming a generic instantiation $\texttt{Ag} \rightarrow ag$.

Then, the loop over kv in Line 7 of Algorithm 1 proceeds as follows:

Iteration	
1	$\mathcal{C} = \{\sim \texttt{can}(ag, ac)\}$
2	$f = \texttt{can}(ag, ac)$ and $\sim f \in \mathcal{C} \rightarrow \mathcal{C} = \{\sim \texttt{can}(ag, ac)\}$

Finally, $\sim \texttt{can}(ag, ac)$ is erased from $\mathcal{C}$ as it is a literal preceded by $\sim$. In the final output set, $\texttt{can}(ag, ac)$ is not included. Consequently, the constraint expressed by the rule with priority 1 has effectively removed action `ac` from the possible action available to an agent with some role `role`. This is indeed equivalent to a regimented prohibition.

3.2. Processing of control rules

Control rules have quite different syntax with respect to the other types, hence their consequences are processed by a different function, Get-Control-Conseqs (see Algorithm 2 in Appendix A). The added difficulty arises from the fact that possibly joint actions have, in general, stochastic consequences. In turn, every potential consequence does not just correspond to a single literal, but to several. Thus, it is not enough to simply return a set of ground literals, but rather a set of potential next states and a probability distribution over those, where every state is characterized by a set of fluents.

As control rules regulate how the state of the system transitions due to the actions performed by the agents, the pre-transition state $s_t = \{f_1, ..., f_n\}$ as well as the joint action whose execution we are examining $\mu = \{\texttt{does}(ag_i, ac_i), ...\}$ need to be added to the ASL description $\mathbb{A}$. Additionally, it must hold that for any participant ag performing some action ac_i (i.e. $\exists \texttt{does}(ag, ac_i) \in \mu$), then ag cannot be taking any other action simultaneously (i.e. $\nexists \texttt{does}(ag, ac_j)$ for any $ac_j \neq ac_i$).

The processing of control rule consequences starts off just as that of the other rule types. First, the instantiations of activated control rules are stored, together with their priority as a key (excluding those exceeding some input threshold) and sorted by descending priority (Lines 3-5). The set of potential next states $\mathcal{S}_{t+1}$ is initialized as a set containing only the empty set and unity probability assigned to the empty set (Lines 6-7).

Then, the function loops over the activated rules (Line 8). Every rule is composed of a probability distribution over joint consequences, which in turn contain several fluents concatenated by the `and` operator. The following check is performed (Lines 9-12): if any of the literals in any of the potential joint consequences is found to be incompatible with any literal in any of the provisional next states in $\mathcal{S}_{t+1}$, then that rule, despite having been activated, is ignored. This is done in order to avoid inconsistencies in the final probability distribution over the post-transition states. This check guarantees that the final probability distribution over $\mathcal{S}_{t+1}$ is correct, i.e. adds up to unity, provided that every control rule also has a proper probability distribution over its set of consequences.

If an activated control rule passes the check, its consequences are added to the set of potential next states (Lines 13-19). Every provisional post-transition state $s_{t+1} \in \mathcal{S}_{t+1}$ has its set of literals expanded with each of the joint consequences of the control rule being examined (Lines 16-17). The probability of the new expanded state is updated as the product between the probability of s_{t+1} prior to expansion times that of the joint consequences, provided by the active control rule (Line 18).

One final step is performed before returning the post-transition states and their probability distribution. A loop is run over the pairs of pre-transition state literals and the post-transition states (Line 20). If a pre-transition state literal is found to be compatible with the literals in the provisional post-transition state, it is dragged over and added to the potential next state (Line 21). When this loop is complete, the set of post-transition states $\mathcal{S}_{t+1}$ alongside with their probability distribution is returned. Note that, because of this final step, if we were to call Get-Control-Conseqs but no control rules were activated, the function would just return the pre-transition state ($\mathcal{S}_{t+1} = \{s_t\}$) with unit probability ($\mathcal{P}(s_t) = 1$).

The use of `incompatible/2` clauses together with the dragging of pre-transition state literals is the approach that ASL takes to tackle one of the problems that any action formalism has to inevitably address: the *frame problem* [37]. The frame problem states that when the state of an action is axiomatized, it should not be necessary to refer to the facts that are not affected by it. In ASL, control rules need only include the facts that do change in their `Consequence` field. By introducing the `incompatible/2` predicate symbol and having the rule interpreter drag the old ground literals that are consistent with those newly derived, the ASL allows for natural expression of control rules where only the facts affected by the actions in the `Condition` field need to be stated.

This approach to the frame problem that ASL takes is totally different from that taken by GDL. In GDL, given a state characterized by a set of fluents $s_t = \{f_1, ..., f_n\}$, all of the post-transition state literals have to be explicitly derived from `next/1` clauses (which, as explained in Section 2.3, are analogous to ASL's control rules). GDL does not make use of a predicate symbol analogous to ASL's `incompatible/2`, and therefore every fact that holds true after a transition from state s_t to state s_{t+1} has to be explicitly stated.

In the discussion of the previous Section 3.1 we introduced the notion of *soundness* for boundary, position and choice rules. An analogous concept applies to control rules. However, due to the more general nature of control rules in comparison with the other types (i.e. the literals in their `Consequence` can be anything), the soundness of control rules relies not on the direct comparison of their consequence literals, but on the use of the `incompatible/2` clauses.

To illustrate the notion of soundness in control rules, picture the following:

```
rule(...,control,N1,if does(Ag1,act1) and does(Ag2,act2) then Conseq1 ...).
rule(...,control,N2,if does(Ag1,act1) then Conseq2 where ...).
```

where `Conseq1` and `Conseq2` are lists of conjunctions of literals with a well-formed probability attached to each of them. The `Constraints` field has been omitted, and for the sake of discussion, assume it is equal for the two rules. Hence, since the conditions of the first rule imply those of the second, if the first rule is activated, then necessarily the second will be activated too.

When the two rules are triggered, three possible scenarios arise. First, the two rules may have compatible consequences. In this case, all the literals that appear in the consequence of one rule are compatible with all the literals in the consequences of the other. In this case, regardless of their priorities `N1` and `N2`, both rules will be processed by Get-Control-Conseqs.

Second, the two rules might not have compatible consequences, but the priority of one of them is strictly larger than the other ($\mathtt{N}_i > \mathtt{N}_j$). Then, the rule with the larger priority prevails, and the other is effectively discarded by Get-Control-Conseqs. The two scenarios discussed so far present no problem and are examples of *sound* control rules.

The third case corresponds to the different control rules having identical priorities and incompatibilities in their consequences. In this case, one of the two rules will prevail and the other discarded. However, it is not straightforward to predict which one will overcome the other, as that depends on the order in which they appear in the rule base Ω and/or the tie-breaking mechanism of the sorting subroutine in Line 5 of Algorithm 2. This is the problematic situation which we refer to as an *unsound* set of control rules.

We close this section by illustrating how the interpretation of control rules by Get-Control-Conseqs enforces the non-regimented norms presented in Section 2. Again, we illustrate punishment to deter participants from executing a forbidden action. The procedure for non-regimented obligations is analogous. Consider the two following control rules:

```
rule(...,control,0,if does(Ag,act) then [default1 withProb D1, ..., defaultN withProb DN] where
                                        []).
rule(...,control,1,if does(Ag,act) then [default1 withProb D1*(1-P), ..., defaultN withProb DN
                                *(1-P), Punish withProb P] where []).
```

The first rule expresses the effects on a generic action by some agent. In general, such an action is non-deterministic, with a set of default consequences each assigned a probability. To simplify the discussion, we leave the `Constraints` field as blank.

The second rule expresses the non-regimented prohibition of the generic action by a higher-priority control rule. Now, the probability of incurring punishment corresponds to P, while the probability of being undetected stands at $1 - P$. The relative proportion of the probabilities for the default consequences is preserved in the prohibiting rule with respect to the default one.

Additionally, consider the following set of `incompatible/2` clauses:

```
incompatible(default1, Punish).
        ⋮
incompatible(defaultN, Punish).
```

When Get-Control-Conseqs is called on an ASL description containing such rules (with argument *threshold* ≥ 1, and assuming that the conditions are such to trigger the rules under discussion), the execution of the algorithm up to Line 7 (where the activated control rules are gathered), results in the kv local variable:

$$kv \leftarrow [\ 1\text{: [}\mathtt{default1\ withProb}\ D_1 * (1-P), ..., \mathtt{default1\ withProb}\ D_N * (1-P), \mathtt{Punish\ withProb}\ P\mathtt{]},$$
$$0\text{: [}\mathtt{default1\ withProb}\ D_1, ..., \mathtt{defaultN\ withProb}\ D_N\mathtt{]}\ ]$$

Then, the main loop in GET-CONTROL-CONSEQS (Lines 8 to 19 in Algorithm 2) first processes the first element in kv, i.e. the consequences derived from the rule with priority 1. This step results in the following post-transition states with probabilities:

$$\begin{gathered}\mathcal{S}_{t+1} = \{\{\texttt{default1}\}, ..., \{\texttt{defaultN}\}, \texttt{Punish}\} \\ \mathcal{P}(\{\texttt{default1}\}) = D_1 * (1 - P) \\ \vdots \\ \mathcal{P}(\{\texttt{defaultN}\}) = D_N * (1 - P) \\ \mathcal{P}(\texttt{Punish}) = P\end{gathered}$$

By the time the consequences derived from the control rule with priority 0 are processed, the incompatibility check in Lines 9 to 12 of Algorithm 2 returns true and discards the consequences derived from the default rule. Hence (and omitting the final steps of the algorithm, Lines 20-21), the returned post-transition states and their probabilities are the ones outlined above, where the punishment consequence is assigned some non-zero probability. The forbidden action is still a possibility by the acting agent (as the action is not banned by choice rules), but its execution might lead to the default consequences or to a deterring punishment.

4. ASL semantics

As reiterated throughout the text, an action situation description has its semantics grounded as an extensive-form game (EFG) with a restricted use of imperfect information. Furthermore, the formal game representation is augmented with a set of literals at those tree nodes that directly correspond to *states* of the system. In order not to get ahead of ourselves, we first need to review some common definitions from game theory and complement them with some definitions unique to this work. We then take a deep dive at how a formal game is built from an arbitrary action situation description, and which properties it is ensured to have as a consequence of the building mechanism. Finally, we present the complexity all the algorithms necessary to build the EFGs, included those introduced in Section 3.

4.1. Background on game theory

In the field of game theory, the simplest model of a multiagent interaction is provided by a normal-form game:

Definition 2 *(normal-form game).* A *normal-form game* is a tuple:

$$\text{NFG} = (P, (A_i)_{i \in P}, (U_i)_{i \in P})$$

where:

- P is the set of *players* (or agents).
- A_i is the set of *actions* available to player i.
- $U_i : A \to \mathbb{R}$ is the *utility function* for player i, which maps every possible joint action profile in $A = \bigtimes_{i \in P} A_i$ to a numeric quantity.

Normal-form games have been widely studied in a variety of fields, from microeconomics to evolutionary theory. However, they are not suitable to capture sequential interactions where agents might take several actions at different times. For this reason, normal-form games are sometimes referred to as stateless games. Moreover, normal-form games do not explicitly store information on possible random effects of joint actions.

To address these shortcomings, it is necessary to work with extensive-form games, where the strategic interaction is represented as a tree that agents transverse as they take actions. To account for stochastic effects, a new artificial player p_0 is added to the set of players, which is usually referred to as "nature" or "chance". We follow the notation by [38]:

Definition 3 *(extensive-form game).* An *extensive-form game* is a tuple:

$$\text{EFG} = (P, (X, E), T, W, \mathscr{A}, p, U)$$

where:

- P is the set of *players* (or *agents*).
- (X, E) is the *game tree*, where X is the set of nodes (or vertices) and $E \subseteq X \times X$ is the set of directed edges. One of the nodes $x_0 \in X$ is the *root* of the tree (with no incoming edges), such that for all other nodes $x \in X \setminus \{x_0\}$ there is a unique path from x_0 to x. The subset of nodes $Z \subseteq X$ with no outgoing edges are called the *terminal* (or *leaf*) nodes.

- $T : X \setminus Z \to P \cup \{p_0\}$ is the *turn function*, which assigns every non-terminal node to the player responsible for taking an action at that node, including chance moves. The turn function induces a partition $\Theta = \{\Theta_0, \Theta_1, ..., \Theta_{|P|}\}$ over the non-terminal nodes, by sorting them into subsets according to the player whose turn it is:

$$\Theta_i = \{x \in X \setminus Z \mid T(x) = i\}$$

- $W = \{W_i\}_{i \in P}$ is the *information partition* which, for every player $i \in P$, W_i corresponds to a partition of Θ_i. W_i splits all the nodes where i makes a move into mutually exclusive sets. Every $w \in W_i$ is called an *information set*. When player i makes a move, the information and actions available to i are exactly the same at any of the nodes that belong to the same information set.
  A game where all information sets are singletons is called *a perfect information game*. In this case, a player always knows precisely what state they are in before making the move. When that is not the case, games are said to have *imperfect information*. The information that is available to a player at a given information set is totally determined by the particular game being analyzed. Although information partitions are typically interpreted as partial observability, they can also be employed to simulate simultaneous moves. The prototype for this case can be found in the extensive form of the one-shot Prisoner's Dilemma game (see Fig. 4).
- $\mathscr{A} = (A(w))_{w \in W}$ denotes the *actions* available to the players, where $A(w)$ is the set of actions available at information set w for the player whose turn it is to move at that set. Actions label the outgoing edges from a decision (non-chance) node.
- p is a function that assigns to every chance node $x \in \Theta_0$ a *probability distribution* over its outgoing edges. Hence, it describes the nature of the environment and its stochastic effects.
- $U = (U_i)_{i \in P}$ is the *utility* function which assigns, for every agent, a numerical payoff for every terminal node, $U_i : Z \to \mathbb{R}$.

EFGs provide much richer representations of multiagent interactions than normal-form games. To generate the semantics of an ASL description, we will use a restricted form of EFGs to model all the possible ways by which a state s_t can evolve given the actions available to the participants at that state. We name this restricted form of EFGs as *game rounds*:

Definition 4 *(game round).* A *game round* is an extensive-form game with the following characteristics:

1. The root node is never a chance node.
2. There is, at most, one information set per player, $W_i = \{\Theta_i\}$, $\forall i \in P$ (players that do not take any action in a particular game round have an empty information partition).
3. 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.
4. If $T(x) = p_0$ (x is a chance node), then all of its child nodes are terminal.

In practice, by condition 2 in Definition 4, in a game round every player has the opportunity to take only one action. Every depth level corresponds to the information set of one player (excluding the terminal nodes and possibly their immediate parents, which might be chance nodes). In a game round, every path of play corresponds to a joint action (i.e. every player only has the chance to select and perform one action), and the sequence of players whose turn it is to take is constant across paths. If the joint action has deterministic results, then the decision vertices are succeeded by a leaf node. Otherwise, if the joint action has stochastic effects, then a chance node succeeds the last decision node, before leading to a leaf node. Note that condition 3 ensures that game rounds always have *perfect recall*. This requirement will greatly facilitate the equilibria computations later on.

For example, the game in Fig. 4b holds the requirements to be a game round. Pointing this example raises the following question: why have we not chosen to model the interaction at a single time-step as a normal-form game, instead of going to the extent of using the much more loaded extensive form and then restricting it?

The answer is motivated by the ability of EFGs to explicitly store the information on probabilistic effects. This is not possible in normal-form games, where every action profile is mapped to a single payoff vector. Consider a joint action profile that leads to several reward allocations, each with some non-zero probability. In the normal-form representation, the payoff assigned to such an action profile would correspond to the weighted average over all the potential rewards, hence losing all information concerning the probability distribution over the outcomes. In our opinion, the loss of this information rules out normal-form games as suitable representations for action situations.

Given a state of the world, a game round models all the possible ways by which, through a single action each, agents might alter the state of the system. Since we want to model relatively complex action situations, with agents executing actions not just once but multiple times, several game rounds will be concatenated and merged into a larger extensive-form game. Next, we explain how game rounds are built from an action situation description and merged into an extensive-form game that grounds the semantics of an ASL description.

4.2. From action situation descriptions to games

Now, we move on to the process that takes an ASL description as input and constructs its game model semantics. This process is performed by the sequential generation of single game rounds that are concatenated into a larger game structure.

First, we address how a single game round is built by the function Build-Game-Round (see Algorithm 3 in Appendix A). It starts off with the set of facts that characterize the pre-transition state s_t, which corresponds to the root of the game round tree. Then, the game tree is built in a *breadth-first* manner, by iterating over the agents that are able to take some action at s_t according to the choice rules (Lines 4-5 and 8-17). One information set is added for each of them (Line 9). At every iteration of this loop, the depth of the game round tree is increased by one level (Lines 11-16).

Once the tree skeleton has been built, it is time to find out which fluents hold true at the potential post-transition states $\mathcal{S}_{t+1}$ (Lines 19-32). Every terminal node corresponds to a different joint action executed from s_t, $\mu = \{\texttt{does}(ag_1, ac_1), ...\}$. Then, for every terminal node the joint action that leads to it from the root node is retrieved and added to the action situation description $\mathbb{A}$ (Lines 20-21). Next, Get-Control-Conseqs is called in order to find out the post-transition states that may be reached from that action profile (Line 23). In case there are no stochastic effects ($|\mathcal{S}_{t+1}| = 1$), the fluents of the single next state are assigned to the terminal node (Line 24). Otherwise, the terminal node is converted into a chance node, and additional descendants are added, one for every potential next state $s_{t+1} \in \mathcal{S}_{t+1}$ (Lines 25-31). The fluents at these new terminal nodes and the probabilities of the edge from the parent chance node are computed by Get-Control-Conseqs. Finally, the joint actions are deleted from the database before moving on to the next terminal node (Line 32).

By construction, the extensive-form game returned by Build-Game-Round fulfills the properties of a game round according to Definition 4. Note that in every game round built by this function, only the root and terminal nodes correspond to actual states of the system, and they are assigned the fluents that hold true at that state (in fact the fluents that hold true at the root node are assigned before the tree emanating from it is constructed). The intermediate nodes between the root and the leafs do not correspond to actual states of the system. Hence, they are not assigned any fluents. They are auxiliary nodes, whose function is to capture the simultaneous nature of joint moves and the possibility for random effects in the environment.

Build-Game-Round manages one last important point. For every joint action profile that is available at s_t, it checks whether executing joint action μ from s_t leads to termination (Line 22). If that is the case, the terminal node that corresponds to the execution of μ in state s_t will not be considered for further expansion (i.e. the construction of the game rounds that emanates from it) when the complete game for the action situation semantics is built.

Now that we know how to model the possible ways by which a state of the system s_t might evolve, the only step that is left is to concatenate multiple rounds into a larger game tree. This is precisely what the function Build-Full-Game does (see Algorithm 4). It only needs an action situation description as data, and maintains a queue of states susceptible to be expanded by Build-Game-Round. The queue is initialized with the initial state of the interaction (Line 12), which is derived as the instantiations of the query `initially(F)` (Line 7). This is done *after* the participants and their roles have been added to the action situation description according to the boundary (Line 2) and position (Line 4) rules respectively, by calling Get-Simple-Conseqs with the appropriate *type* argument. As new game rounds are built, their respective terminal nodes are pushed to the queue as long as the joint actions leading to them from the root node in their game round do not trigger termination (Lines 28-30).

Additionally, every time a state is popped from the queue (Line 14) and its fluents added to the ASL description (Line 17), Build-Full-Game checks whether the termination conditions hold, *prior to the execution of any action* (Line 18). Adding this check to the one performed in Build-Game-Round shows that, in an ASL description, a state may lead to termination either because the facts that characterize it command it, or because of the actions that have led to it from its pre-transition state.

Also, Build-Full-Game takes in a *max* argument that controls the maximum depth of the resulting game tree. The motivation for including this argument is as a safety stop in case the conditions for `terminal/2` clauses to return true are not met at all paths emanating from the root of the tree. If the action situation description is written in a such a way that the above situation arises, having a maximum allowed depth for the game tree being grown ensures termination of Build-Full-Game. Users confident in their ability to write action situation description where termination will always be dictated by `terminal/2` clauses returning true and do not wish for their EFGs representations to be cut short can use argument $max \rightarrow \infty$.

Finally, Build-Full-Game returns the extensive-form game that represents the action situation description, augmented by the facts that characterize the nodes that directly correspond to states of the system (those that are the root of game rounds plus the leafs). Hence, we can define the semantics of a valid ASL description by construction:

Definition 5 *(ASL semantics).* The *semantics of a valid ASL description* $\mathbb{A}$ correspond to an extensive-form game Γ with a restricted use of imperfect information, and a function $\mathcal{F}$ over a subset of the nodes in Γ. Γ and $\mathcal{F}$ are computed exactly by Build-Full-Game. Γ is built as the concatenation of game rounds, and is augmented with $\mathcal{F}$, which assigns a set of fluents for every node in Γ that correspond to a proper state of the system.

It should be noted that Build-Full-Game does not deal with the *utilities* assigned to the leaf nodes of the game tree. We choose not to assign utilities at construction time in order to allow flexibility in the valuations that agents make of

outcomes. If one is conducting a classical game-theoretical analysis based on material or monetary rewards, a predicate standing for such variable can be introduced and its evolution modeled through control rules, as stated in Section 2.1. Then, for every agent, the utility at a terminal node is extracted from the `payoff(Ag,x)` fluents assigned to that node. Alternatively, if one wishes to model other values that the agents might take into consideration in the particular action situation, the utilities can be assigned as a function of the fluents that hold true at every terminal node and/or the path of play from the root to the terminal node.

4.3. From games to action situation descriptions

Definition 5 specifies the semantics of any action situation description $\mathbb{A}$ as an EFG that respects the validity conditions outlined in Definition 1. To show the expressive power of the Action Situation Language, we now prove the equivalence between valid action situation description and extensive-form games that can be identified as a concatenation of game rounds. Despite that the focus of this work is on the transformation from an action situation description to an extensive-form game, we discuss here the inverse process for completeness purposes.

In order to continue, we must first clarify when an EFG is identifiable as a concatenation of game rounds. Consider an EFG Γ, and any of its subgames γ.[2] Now, take the largest subgame within γ, if any, and replace it by a terminal node. If after this transformation is applied to any of the subgames in Γ, it fulfills the conditions for a game round in Definition 4, then the overall game Γ is identified as a concatenation of game rounds.

We are now ready to establish the equivalence between EFGs and action situation descriptions:

Theorem 1. *An extensive-form game Γ is a concatenation of game rounds if and only if there exists an action situation description $\mathbb{A}$ such that $\mathbb{A}$ has its semantics grounded as Γ by* Build-Full-Game.

Proof. The reverse implication follows directly by construction from Definition 5. For the forward implication, we construct an action situation description *ad hoc*, and then prove that it certainly has its semantics grounded as the game in question.

Given Γ, start by having all of its players as agents: add `agent(p)` to the agent base Δ, $\forall p \in P$. Second, all agents have to be admitted as participants of the action situation and assigned a distinct role. Although in some cases (e.g. the Prisoner's Dilemma example in Section 2.3) several agents are designated the same role, for the sake of generality we assign a distinct role the every agent, named after themselves. Hence, the following two rules per participant are added to Ω:

```
rule(id,boundary,0,if agent(P) then participates(P) where []).
rule(id,position,0,if participates(P) then role(P,P) where []).
```

Third, denote the initial state as s_0 and denote all states as incompatible with one another. To do so, write the predicates `initially(`s_0`)` and `incompatible(_,L) :- length(L,1)` to the states base Σ (i.e. only one fluent s_i per state).

In the worst case, the input EFG Γ does not contain any regularities whatsoever, and so dedicated choice and control have to be added for every game round. To do so, it is necessary to first identify the chunks of the game tree that correspond to game rounds. Then, the dedicated rules need to be written for each of them.

The checking procedure discussed previously to identify whether an EFG corresponds to a concatenation of game rounds can be used to identify them. Take any subgame of the original extensive-form game, and replace its largest subgames into terminal nodes. The result corresponds to a game round.

Once every game round is delimited, denote its root as state s_i. For every player that takes some action within that game round, write a set of choice rules that assign to it the available actions, following the template:

```
rule(id,choice,0,if role(P,p) then can(P,a^p_ij) where [s_i]).
```

where a^p_{ij} denotes the j-th action that player p can take in state s_i.

The rule base Ω is completed by adding the control rules. In the worst case, one control rule has to be added for every *joint* action profile that players can take in every game round. Given a chunk of the overall game tree that has been identified as a game round, traverse it until either a chance node or a terminal node is encountered, whichever is first. Each distinct traversal will lead to a new control rule. Consider one of these paths from the root to either a chance or a terminal node in the game round. For every decision node x, add the literal `does(`$T(x), act$`)` to the `Condition` field of the control rule, where $T(x)$ is the turn function at node x and act is the action label of the outgoing edge. If the path ends in a terminal (non-chance) node, set the `Consequence` field to `[`s_{i+k} `withProb 1]` ($k = 1, 2...$). Otherwise, if a chance node is encountered, for every of its outgoing edges, append `[`s_{i+k+m} `withProb` p_m`]` $m = 1, 2...$ to the `Consequence` list. Finally, set `Constraint` to `[`s_i`]`, to indicate that the control rules only apply to the game round emanating from node s_i. Last of all, the individual terminal nodes of the overall game tree have to be designated as such, by adding `terminal :-` s_T clauses to Σ, one per terminal node.

[2] For a definition of subgame of an EFG, see [39, p.45].

Table 3
Time complexity of the algorithms presented in Appendix A.

Algorithm	Complexity
Get-Simple-Conseqs(..., *type*="boundary", ...)	$\mathcal{O}(R_b \cdot \log R_b)$
Get-Simple-Conseqs(..., *type*="position", ...)	$\mathcal{O}(R_p \cdot \log R_p)$
Get-Simple-Conseqs(..., *type*="choice", ...)	$\mathcal{O}(R_{ch} \cdot \log R_{ch})$
Get-Control-Conseqs	$\mathcal{O}(C^{R_{cnt}} \cdot (C \cdot L + F))$
Build-Game-Round	$\mathcal{O}((R_{ch})^{Ag} \cdot C^{R_{cnt}} \cdot (C \cdot L + F))$
Build-Full-Game	$\mathcal{O}((R_{ch})^{(max+1)Ag} \cdot C^{(max+1)R_{cnt}} \cdot (C \cdot L + F))$

To prove that the *ad hoc* ASL description constructed above has its semantics grounded by Build-Full-Game as the original EFG Γ, we combine empirical and induction arguments. For an EFG that consists of $n = 1$ game rounds, we experimentally prove that such an action situation description has its semantics correctly grounded.[3] For a larger game with $n > 1$ game rounds, the subsequent game rounds are also individually correctly built, by the results obtained for $n = 1$. The subsequent game rounds are appended to the first one (whose root equals to the root node of the overall game tree) by changing the i sub-index in the action and state atoms (a^P_{ij} and s_i respectively) of the choice and control rules. □

The construction procedure outlined above is very tedious and results in a large ASL description (i.e. with a large amount of `rule/4`), even for small extensive-form games. However, it is generally expected that the extensive-form game will contain some symmetries or regularities that will avoid the need for a set of distinct choice and control rules per every game round.

In their original formulation, EFGs do not carry information on their nodes. However, according to the *ad hoc* ASL description above and the Build-Full-Game function, the resulting EFG will be expanded with a generic state fluent function $\mathcal{F}$ that assigns a generic state fluent s_i to those nodes that correspond to roots of game rounds. Also, in order to completely recover the original game, it is necessary to assign *a posteriori* the utilities to its terminal nodes. However, none of these two minor differences between the input EFG and the semantics grounded from the constructed ASL description are related to the structure of the game, i.e. the topology of the game tree.

4.4. Game construction complexity

Now that all the algorithms in this work have been reviewed, we discuss the complexity of grounding the semantics of an action situation description as an extensive-form game. Complexity results for the algorithms included in this work are presented in Table 3, which follows the following notation:

- $R_{b/p/ch/cnt}$: number of boundary/position/choice/control rules.
- C: maximum number of consequences per control rule (i.e. length of the `Consequences` list in control rules).
- L: maximum number of literals per consequence in control rules.
- F: maximum number of fluents that describe a state.
- Ag: number of agents in the agent base Δ.
- *max*: maximum allowed depth of the game tree, passed as a parameter to Build-Full-Game.

The complexity of Get-Simple-Conseqs is dominated by the sorting procedure in line 5 of Algorithm 1. Assuming that the Merge-Sort algorithm is employed, the complexity is $R \log R$, where R is the number of activated boundary, position and choice rules, which is assumed to be equal to the total number of rules of each type in the worst case. For the other algorithms, the complexity is dominated by the iterations over activated control rules, the number of consequences in their `Consequence` field C and the number of literals in every consequence L. To derive the complexity of Build-Game-Round and Build-Full-Game, we assume that the number of actions available to every agent equals the number of choice rules in the ASL description.

The most important function for the purposes of this work is Build-Full-Game, as it is called once per every action situation configuration that we wish to assess. The complexity of this algorithm scales exponentially with the number of agents, the number of control rules, and the maximum allowed depth for the resulting EFG. Despite the exponential explosion, Build-Full-Game is well-suited for parallelization. The game rounds at every depth level of the overall game tree can be constructed independently from one another, i.e. the calls to Build-Game-Round made within Build-Full-Game can be distributed over several computing nodes. Furthermore, information sets are, by construction, contained within one game round and do not include nodes belonging to different game rounds. Therefore, information sets do not need to be stored during computation and the construction of the overall EFG can be efficiently parallelized.

[3] See the `examples/adhoc` directory of the ASL distribution.

```
1 rule(ipd,choice,1,if role(P,prisoner) then ~can(P,defect) where [consecutiveDefections(P,N),N
      >=2]).
2 rule(ipd,control,1,if does(P,defect) then [consecutiveDefections(P,M) withProb 1] where [
      consecutiveDefections(P,N),{M=N+1}]).
3 rule(ipd,control,1,if does(P,cooperate) then [consecutiveDefections(P,0) withProb 1] where []).
4
5 initially(consecutiveDefections(P,0)) :- role(P,prisoner).
6 incompatible(consecutiveDefections(P,_),L) :- member(consecutiveDefections(P,_),L).
7
8 % rules to change the outcome of mutual defection
9 rule(ipd,choice,2,if role(P,prisoner) then can(P,defect) where []).
10 rule(ipd,control,2,if does(P1,defect) and does(P2,defect)
11     then [payoff(P1,Y11) and payoff(P2,Y12) withProb 0.5,
12           payoff(P1,Y21) and payoff(P2,Y22) withProb 0.5]
13     where [P1@<P2,payoff(P1,X1),payoff(P2,X2),{Y11=X1+0,Y12=X2+9,Y21=X1+9,Y22=X2+0}]).
```

Listing 6: Additional regulations with priority 1 to limit the number of consecutive defections (top), and with priority 2 to change the outcome of the mutual defection (bottom).

In the best-case scenario, there are more processing units than the maximum number of game rounds that have to be built at any depth level of the game tree. In such a case, the time complexity of Build-Full-Game would be reduced to that of Build-Game-Round, plus some communication overhead (i.e. the state fluents of a terminal node have to be transmitted to the host in charge of expanding the game round emanating from that node).

As an example, take the iterated Prisoner's Dilemma first introduced in Section 2.3. Its action situation description has the following parameter assignments: $R_b = R_p = 1$, $R_{ch} = 2$, $R_{cnt} = 4$, $C = 1$, $L = 2$, $F = 3$, $Ag = 2$ and $max = 3$. Hence, the complexity of Build-Full-Game for this ASL description is $\mathcal{O}(5 \cdot 2^8)$. If parallelization infrastructure is available and there are enough processing units ("enough" in this case meaning at least 16, which is the number of game rounds at the broadest level), then the complexity is reduced to that of Build-Game-Round, $\mathcal{O}(5 \cdot 2^2)$, a reduction by a factor of 2^6.

4.5. Follow-up on the iterated Prisoner's Dilemma

As an example of ASL semantics generation, we go back to the iterated Prisoner's Dilemma example presented earlier. The extensive form game built from the default rules presented there is shown in Fig. B.6 (B.1). The utilities that appear below the terminal nodes have been set by the `payoff(Agent,X)` fluents assigned to it.

To illustrate the introduction of higher priority rules and the impact that these have on the semantics, we propose two examples. First, we consider a ban on the number of consecutive defections that agents can take. Second, we introduce random changes to the outcome resulting from both agents playing "defect", where the outcome of the "defect"-"defect" joint action is as if one participant had defected while the other kept cooperating. The cheater and the sucker rewards are assigned randomly to the two agents. The intuition behind the first set of higher-priority rule is fairly straightforward, as it is a simple example of regimented norm (in this case a prohibition) to avoid the executing of too many actions considered to be detrimental.

The intuition behind the second set of higher-priority rules is not so obvious. To illustrate a possible real-world example that is captured by such rules, consider the following scenario. International talks to implement pollution-reduction policy to fight climate change (and the relatively little progress made during such talks) have often been identified as a Prisoner's Dilemma-type situation [40]. Consider two nations with competing economies that could introduce new regulations to reduce emissions, at an expense to their economic output. If they both, simultaneously, cooperate (i.e. transform their economies to reduce emissions), they will both be better off in the long run. If one defects (i.e. keeps polluting) while the other introduces environmental policy, the cooperator is expected to suffer a loss in competitiveness and therefore a hit to their economy, which the polluter takes advantage of. Finally, if both countries avoid emission reduction, the pollution levels will result in extreme weather events at an unmanageable frequency. Such events can partly destroy infrastructure, hence highly hindering the economic capacity of the location hit by them and providing an advantage to the country that has not suffered it, who can expect to sell their goods at a higher price. However, such events will, most likely, only hit infrastructure in one of the two countries at a time. Assuming that the two countries are equally likely to suffer from extreme weather events, this situation can be encapsulated by a random change in the "defect"-"defect" (or, in this case "keep polluting"-"keep polluting") outcome.

The additional rules needed to introduce those regulations appear in Listing 6. At the top, in order to limit the number of consecutive defections, it is enough to introduce one new choice rule with priority 1 that turns the action "defect" unavailable if the agent in question has defected twice in a row. This is an example of a prohibition rule in the regimented modality. Additionally, there are two new control rules that control the evolution of the `consecutiveDefections/2` literals, as well as their initialization and the consideration that there may only be one `consecutiveDefections/2`

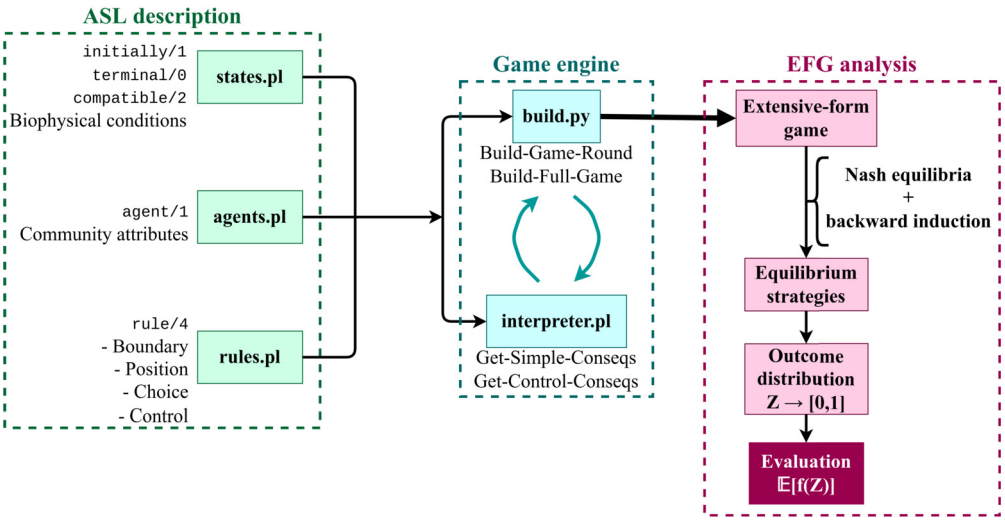


Fig. 5. Implementation of the computational model of the IAD framework.

literal per agent. The EFG that results from the interaction of this rule appears in Fig. B.7. Now, at some game rounds some agents only have action "cooperate" at their disposal, and the symmetry of the game tree is broken.

At the bottom of Listing 6, the rules that change the outcome of the mutual defection are introduced with priority 2. First, they need to undo the ban on several consecutive defections through a choice rule that recovers the action "defect" under any circumstances. Then, we present the control rule that is most representative of this regulation. Now, if both agents defect, the resulting outcome is *as if* one agent had cheated on the other. The selection for who becomes the *de facto* cheater and sucker is done by flipping an unbiased coin. The resulting game semantics for this configuration are displayed in Fig. B.8.

5. Implementation and computation of equilibria

Now that we have extensively presented the formal syntax and semantics of ASL, we go into the practical aspects of writing an action situation description, generating the EFG that derives from it and evaluating its structure.

The complete set-up to perform the *what-if* analysis of a rule configuration is displayed in Fig. 5. First, we write the ASL description with its clauses divided according to the separation $\mathbb{A} = \Delta \cup \Sigma \cup \Omega$ into agents, state-related information, and rules into three corresponding files states.pl, agents.pl and rules.pl.

These three files are fed to the game engine, that consists of the rule interpreter plus the game builder. The first is a Prolog script directly responsible for querying and processing the activated rules, and contains the implementations of Get-Simple-Conseqs and Get-Control-Conseqs. The second is a Python script that repeatedly communicates with the rule interpreter[4] and generates the action situation semantics through its implementation of the Build-Game-Round and Build-Full-Game functions.

Once the EFG is generated and utilities assigned to its terminal nodes, an assessment of its incentive structure can be performed with standard game-theoretical tools. In particular, the construction of EFGs as a concatenation of game rounds greatly facilitates their integration with *sequential rationality* solution concepts. Note that every game round is ensured to have perfect recall. Consequently, the resulting game tree also has perfect recall. A classic result in game theory by [41] establishes that in imperfect-information EFGs of perfect recall, for every *behavioral* strategy there is a corresponding equivalent *mixed* strategy, and vice-versa. In game theory, a behavioral strategy over an EFG for player i fixes, for every information set $w \in W_i$, a probability distribution over the available actions at w. Meanwhile, a mixed strategy for player i corresponds to a probability distribution over the sequences of actions that player i takes as they transverse the game tree.

Considering this result, we propose to use *backward induction* to compute the equilibrium strategies in an EFG generated from an ASL description. The general procedure operates as follows [39, p.46]:

1. Identify the *final* subgames of the overall EFG, i.e. those that do not have subgames of their own nested within them. Compute the equilibria strategy (or whichever solution concept) for the final subgames. The most common notion of rationality in game theory is expressed by the Nash equilibrium [42], however other options are possible, such as correlated equilibria [43].

[4] The communication between the Prolog script interpreter.pl and the Python script build.py is realized thanks to the open-source PySwip package.

2. Replace the subgames by terminal nodes whose utilities correspond to the expected average obtained by playing the equilibrium strategies.
3. Repeat the previous steps until the root node the expected utilities at the root node are computed.

There are several particularities of the EFGs we work with in this paper that make the above procedure particularly well-suited to compute the equilibrium strategies in action situations. First, the roots of all subgames of the overall EFG also coincide with the roots of the game rounds that have been concatenated to build it. Additionally, one should first take advantage of the fact that the game rounds that are closest to the leaf nodes of the overall game tree are also the smallest *subgames* within the tree, i.e. they are the *final* subgames of the overall EFG.

Second, these subgames in extensive form can be easily transformed into their induced normal form [44, Ch.5]. This is due to the fact that, by construction, players only make one move in a game round, and they are each assigned an information set relating "sibling" nodes at the same depth level of the game tree. Then, to compute the normal form from the extensive form of a game round, the chance nodes (if any) are substituted by terminal nodes with utility averaged over its children. The mapping from every joint action profile to a utility vector $\langle U_i \rangle_{i \in P}$ is computed by traversing all the paths in the game tree. The joint action profile corresponds to the actions by all players that are encountered along the path, while the utilities are extracted from the terminal nodes (which have originated from an average over the children of a chance node). Note that this procedure to convert an EFG into an NFG is only valid for game rounds fulfilling the properties of Definition 4, and does not generally apply to an arbitrary EFG.

Once the final game round has been converted from extensive into normal form, compute the equilibrium strategies at these normal-form game, using whichever solution concepts one deems appropriate. Finally, substitute the whole game round by a terminal node. Set its utilities to the average of the distribution induced by the previous equilibrium strategies together with the probabilities over chance nodes. Repeat these steps until the root of the original EFG is reached.

Note that the resulting equilibrium strategies computed by this procedure are given as *behavioral strategies*. For every game round, player i is assigned some probability distribution over the actions available to them at the only information set of player i at that game round. Consequently, the backward induction procedure assigns, for every information set in the game tree, one probability distribution for the actions available at that information set. This is precisely the definition of a behavioral strategy. Thus, following [41], we are guaranteed that there is some equivalent mixed strategy for the overall EFG.

Note also that the resulting equilibria strategies are not just rational but also *sequentially rational*. They correspond not just to equilibria over the whole game tree, but the restriction of the computed strategy to every subgame is also an equilibrium over it. This property is referred to as *subgame perfect equilibrium*.

So actually, limiting the use of imperfect information to simultaneous moves (hence ensuring perfect recall) does limit the expressive power of our language, but has major advantages when it comes to computing the equilibria that the game structure incentivizes. First, only "small" subgames have to be converted from the extensive to the normal form, as in every subgame any agent has at most one information set. Although this transformation can lead to exponential blow-up in the general case, we keep this complexity under control since it is single game rounds (where players take just one action) that have to be converted.

Second, instead of needing one big transformation from the whole EFG to a normal-form representation to find the mixed equilibria strategies directly, we can compute the equivalent behavioral strategies by performing several less intensive transformations over the game rounds only. An added advantage is that this approach avoids the contradictions often generated when turning a complex EFG into its normal-form representation. The computation of equilibria on the normal-form representation of a complete EFG may lead to equilibria that, although being rational are not *sequentially* rational, and hence constitute non-credible threats [45].

Once the resulting equilibria strategies are available, it is straightforward to find the probability distribution they induce, together with the probabilities assigned to chance moves, over the terminal nodes Z of the game tree (see the "Outcome distribution" box in Fig. 5). Finally, an evaluation criteria (which we generically denote as a function f over the leaf nodes Z) is defined, and its expected value is computed given the induced outcome distribution. This simple computation concludes the process from a rule configuration (an ASL description) to its evaluation. Now, the communities of agents involved can quantify whether the rules in place are in accordance with their idea of a "good" outcome.

The backward induction procedure outlined previously in this section is included with our ASL distribution. Step 2 of the procedure (i.e. the computation of Nash equilibria at every normal form subgame) is achieved by applying the incentive minimization approach of [44, p. 104]. Also, the computation of distribution over outcomes given the subgame perfect equilibria strategies is also included as part of the code distribution. The distribution over outcomes is computed as the product of the probabilities of all the action in the path from the root to that terminal node (or the probability of nature moves, if there are any). For all the example in this paper, we use the Nash equilibria and the computation of the distribution over outcomes implemented alongside the ASL distribution.

To exemplify the implementation issues presented in this section, we turn to the iterated Prisoner's Dilemma running example. We verify that the different rule configuration we have presented (the default ones in Section 2.3 and the additional

rules in Section 4.5) do indeed encourage different behaviors by the agents and hence lead to different outcomes. These results are presented in Tables B.4 to B.6 for the default rules, the rules limiting the number of consecutive defections and the rules banning the mutual defection, respectively. Tables B.4 to B.6 show, for every rule configuration, the equilibrium strategy induced by the structure of the EFG (obtained as the Nash equilibria) and the probability distribution that these equilibrium strategies induce over the terminal nodes. Note that equilibrium strategies are presented as behavioral strategies (i.e. a probability distribution over the action available at every information set).

As expected, the default Prisoner's Dilemma game has its equilibrium at pure defection by all players at all rounds of the game. However, the introduction of the limit on consecutive defections does encourage some cooperative actions by the participants at some decision points, and hence the outcomes with non-zero probability all lead to higher payoffs for both players. Finally, with the rules implementing the ban on mutual defection, agents go back to defecting at all rounds of the game. However, because of the randomness introduced, several outcomes are equally likely, with unequal distribution of payoffs.

6. Further examples

Before presenting the final conclusions, we illustrate the versatility of the ASL language with two more examples. These are more sophisticated than the very generic Prisoner's Dilemma, and also more interesting from the policy analysis and socio-economics perspective.

6.1. Axelrod's (meta)-norms game

First, we use ASL to model the norms and metanorms games, originally proposed by Robert Axelrod [46].[5] This action situation can be seen as a more elaborated version of the Prisoner's Dilemma. In the *norms* game, an individual i has the opportunity to defect and benefit at the expense of others. If they do not defect, no benefits or costs are incurred by anyone. If they do defect, then there is a fixed probability that they will be detected by a monitor j. If the cheater is detected, then the monitor can choose to punish them at a cost to themselves, but also imposing a large loss to the cheater.

In an extension called the *metanorms* game, the monitor j is themselves being watched by a meta-monitor k. Similarly to the norms game, the meta-monitor may detect with some probability if the monitor has neglected their duty (i.e. has chosen not to sanction i despite detecting their defection). In that case, the meta-monitor may decide whether to punish the monitor or not.

The norms and metanorms games are interesting examples for two reasons. First, they illustrate the use of non-regimented norms, as "bad" actions (defection by an individual or neglect by the monitor) always remain feasible actions. Furthermore, the agency of those responsible for monitoring and sanctioning is explicitly introduced into the game, instead of being idealized away as part of the environment dynamics (as in e.g. [34]).

The rules structuring the *norms* game are designated as the default ones and hence have priority zero. The meta-monitoring is introduced with additional regulations of priority 1. The resulting games and state fluent semantics appear in Fig. B.9 (see Appendix B.2). The norms and metanorms game trees are constructed with BUILD-FULL-GAME using arguments *threshold*=0 and *threshold*=1 respectively. The utilities that appear below the terminal nodes are set from `payoff(Ag,X)` literals, assigning to agent `Ag` the utility value `X`.

The state fluents for the nodes that correspond to actual states of the systems are also shown. For the terminal nodes, the probability distribution over the leaf nodes induced by the equilibrium strategies given the utilities appears next to their fluents. The equilibrium strategies themselves are displayed in Table B.7. These results show that, given the current payoff structure of this action situation, the introduction of a meta-monitor does not have any positive effect. Before k joins the interaction, agent i chooses to defect and the monitor j chooses not to sanction i if detected. However, the addition of k does nothing more that perpetuate "bad" agent behavior, as k chooses not to sanction j if their negligence is detected.

6.2. Ostrom's fishing game

The last example for which we write an ASL description first appeared in [47, Ch. 4].[6] It illustrates a theoretical analysis of how community-crafted rules are capable of transforming the opportunity structure that individuals face in common-pool resource environments. This is the richest example we present of this work, as it illustrates the use of community attributes, non-default position rules and biophysical features.

In this example, two fishers have access to an open-water fishery. Starting at the shore, they may go to one of two fishing spots (one is assumed to be more productive than the other). If, after the first trip, both fishers meet at the same spot, they can choose to stay or leave. If they meet yet again at the same spot after that second action, they inevitably fight. The winner of the fight is determined by flipping a biased coin, whose probabilities depend on the relative strengths of every fisher.

[5] See the `examples/metanorms` directory of the ASL distribution.
[6] See the `examples/fishers` directory of the ASL distribution.

This is the first example that utilizes agent attributes (`speed/2` and `strength/2`). The relative difference between the strength and speed of the two agents is used to compute the probability that one will win a fight or a race over one of the fishing spots. Also, this is the first example that adds biophysical conditions (two fishing spots declared with predicate `fishing_spot/1`). The declaration of fishing spots determines some of the actions that the fishers may take, since they can only leave the shore for one of them. Overall, both the community attributes and the biophysical conditions in this example strongly influence the resulting EFG, however we keep them constant throughout the analysis as they are assumed to be non-changeable in the short to medium term. Also, note how the user is free to choose the predicate symbols (as long as they are not reserved keywords, see Table 1) for community attributes and biophysical conditions. This is the case because, unlike rules, community attributes and biophysical conditions do not require custom querying and interpretation functions.

The boundary and position rules in this action situation are analogous to the other examples, i.e. all agents participate and they all take on the same role, that of *fisher*. Initially, both agents start at the shore. The interaction halts when each fisher is in a different spot, or when a fight has occurred. At every state, one fisher may only be at one place, and there can only be one loser and one winner of fights or races (the introduction of a racing condition is introduced when we introduce higher-order rules).

The choice rules are also very easy to interpret. They state that fishers at the shore can go to any of the fishing spots. Once there, they may choose to stay or leave. As for the control rules, they assume that fishers always get to wherever they intended to go (i.e. boat engines never break down). The last control rule states that, if fishers are at the same fishing spot and they both take the same action, a fight will ensue (since they meet again either at the same spot or at another one). The semantics of this ASL description (using the rules with priority equal to zero) are displayed in Fig. B.10.

In order to avoid violent fights, additional higher priority rules can be introduced. The first option is to implement a *first-in-time, first-in-right* scheme. Now, if fishers depart from the shore with the same destination spot, they race to get there. The first to get there is guaranteed to keep the spot, while the slower fisher has to leave. The winner of the racing contest is not determined by their `strength/2` attribute, but by their `speed/2` instead. The corresponding semantics of this rule configuration appear in Fig. B.11.

Another possibility is the implementation of a *first-to-announce, first-in-right* scheme. In this case, one of the participating agents is randomly assigned to a new role, which we refer to as *announcer*. Before anyone has left the shore, the announcer has to broadcast the spot where they intend to fish. Then, if they hold their promise and go there, they are guaranteed it, i.e. they always win the race to get there. If the announcer goes to a different spot than the one they have broadcast, a race ensues between the two fishers, analogous to the *first-in-time, first-in-right* scheme.

The *first-to-announce, first-in-right* scheme shows the first instance of a non-default position rule. In this case, one randomly chosen fisher is designated as the announcer. Assuming this additional role provides the participant with new actions, as well as conditions the actions of others (e.g. fishers cannot leave the shore before the announcer has announced their fishing spot of choice). Note that the announcer role is in addition to the role of fisher, not in substitution of it. This extra position rule effectively breaks the symmetry that participants previously had, when they both assumed the same roles and hence could execute the same actions. This feature definitely adds to the richness and complexity of the interaction.

Note that rules for this latter configuration have priority 2, since they are added *on top of* the rules for the previous *first-in-time, first-in-right* scheme. This is the first example that illustrates a non-default position rule, such as the one that creates the *announcer* role. The game semantics for this rule configuration appear in Fig. B.12.

The utilities at the terminal nodes of Figs. B.10 to B.12 are set by assigning the following costs and benefits to actions and outcomes: a fisher keeps a spot to themselves or wins the fight over it ($v_1 = 10, v_2 = 5$), a fisher looses a fight ($d = -6$), a fisher travels between spots ($c = -2$). The resulting equilibrium strategies, obtained with an identical computation to the previous examples, appear under the corresponding game trees.

We evaluate the resulting outcome distributions qualitatively. For the default rule configuration, violent outcomes are predicted for over 50% of the paths of play (terminal nodes 14 through 27). Both the *first-in-time, first-in-right* and the *first-to-announce, first-in-right* schemes avoid violence, and hence promote more socially desirable outcomes. However, for the *first-in-time, first-in-right* configuration the fishers still go to the same spot and compete for it. This outcome is avoided in the *first-to-announce, first-in-right* scheme. Now, the announcer prefers to proclaim the most productive spot (spot 1) and remains faithful to their announcement. The other fisher, then, chooses to go for the other spot. Hence, honest announcements are encouraged, and any sort of competition is prevented with the *first-to-announce, first-in-right* scheme.

7. Conclusions

In this work, we have presented a complete computational model of Elinor Ostrom's Institutional Analysis and Development framework. It is based on the Action Situation Language, a novel logical language for the description of action situations, whose friendly syntax is highly tailored to the components identified in the IAD framework. In particular, the IAD framework identifies three sets of exogenous variables that are responsible for shaping social interaction: biophysical conditions, attributes of the community, and rules. The computational model we propose respects this distinction and stores the knowledge corresponding to each of the three exogenous variables separately. In this work, special attention is paid to the *rules* variable, as this is the only one susceptible to changes in the short term. Consequently, the Action Situation Lan-

guage includes a mechanism to solve conflicts between contradicting rules, that allows new rules with higher priority to override older rules with lower priority.

The Action Situation Language is complemented with a game engine that automatically generates the semantics of a description as an extensive-form game. The resulting EFG has some self-imposed limitations, however these greatly facilitate the computation of equilibrium strategies, the distribution over outcomes and their evaluation. We have illustrated the use of ASL and the complete process from an action situation description to the evaluation of its impact with some examples. Notably, the fishers action situation of Section 6.2 demonstrates the complete analysis, where the outcomes are evaluated in terms of the avoidance of violence, competition, and honesty by the participants. Additionally, it shows how the introduction of suitable regulations is able to steer the system towards more desirable end states.

The work presented here has several limitations and is susceptible to extensions in several directions. We point to five potential research directions that may take the work presented in this paper as a starting point:

1. The incorporation of an *information* rule type could greatly enhance the expressive capabilities of ASL, as it could potentially provide the syntax for games with imperfect information (beyond simultaneous moves), or games with imperfect recall. Work in this direction should explore what changes need to be incorporated to the BUILD-FULL-GAME algorithm to include more sophisticated schemes of information accessibility.
2. The *possibility for dynamic boundary and position rules* that are queried not just before the interaction kicks off, but also while it is ongoing. Such a refinement would allow agents to get in and out of an action situation and/or switch roles dynamically.
3. Studies on the *relationship between ASL and logical action formalisms* such as Situation Calculus, to expand the reasoning schemes applicable to an ASL description. For example, how should ASL rule statements be translated into Situation Calculus domain axiomatizations?
4. Work on the formal verification of an action situation description, in terms of its *validity*, *soundness* and *relevance* of the included rules. For example, a position rule assigning a participant to an agent with no actions available has no effect on the outcomes of the interaction. Interesting work could be developed to define this sort of mutual dependencies between rule statements and other components of the action situation description.
5. Studies on the *nesting of action situations*. In this work, we have focused on *operational* action situations, where agents interact directly with one another and their shared environment. However, ASL could also be used to describe *collective-choice* action situations (e.g. a negotiation domain or a voting procedure), whose outcomes result in the implementation of new rule statements on an operational action situation. The linkages between two such ASL descriptions are also worth exploring.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

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) and the VALAWAI project (Horizon #101070930).

Appendix A. Algorithms

For every algorithm pseudo-code, the following is specified:

- **Input:** its arguments.
- **Output:** its return value(s) and/or data structure(s).
- **Data:** the information stored in the database being consulted. In general, all functions call upon the original action situation description $\mathbb{A}$. Additionally, the facts corresponding to the current state s_t and/or the action profile executed μ might also be necessary.
- **Description:** Short documentation on the procedure implemented by the function.

Algorithm 1: Function GET-SIMPLE-CONSEQS(*id*, *type*, *thres*).

Input : *id* ▷ string
type ▷ one of either "boundary", "position" or "choice"
thres ▷ non-negative integer
Output : $\mathcal{C}$ ▷ a set of ground atoms
Data : $\mathbb{A}$ ▷ an action situation description
$\phi = \{\texttt{participates}(ag_i)\}$ ▷ set of participant fluents (if *type*="position")
$\rho = \{\texttt{role}(ag_i, r_i)\}$ ▷ set of role fluents (if *type*="choice")
$s_t = \{f_1, ..., f_n\}$ ▷ set of state fluents (if *type*="choice")
Description : Get the participants, roles or available actions entailed by the boundary, position or choice rules respectively.

Function GET-SIMPLE-CONSEQS(*id, type, thres*)**:**
  $kv \leftarrow []$
  foreach instance of `?-query_rule(rule(`*id*`,`*type*`,`*pr*`,if Cond then Conseq where Constr))` **do**
    if $pr \leq thres$ **then** kv.APPEND(pr : `Conseq`)
  $kv \leftarrow$ SORT-BY-DESCENDING-KEY(kv)
  $\mathcal{C} \leftarrow \{\}$
  for $(pr : f)$ pair in kv **do**
    if $f \notin \mathcal{C}$ **and** $\sim f \notin \mathcal{C}$ **then** $\mathcal{C} \leftarrow \mathcal{C} \cup \{f\}$
  $\mathcal{C} \leftarrow \{c \in \mathcal{C} \mid c \neq \sim f\}$
  return $\mathcal{C}$

Algorithm 2: Function GET-CONTROL-CONSEQS(*id*, *thres*).

Input : *id* ▷ string
thres ▷ non-negative integer
Output : $\mathcal{S}_{t+1} = \{s^1_{t+1}, s^2_{t+1}, ...\}$ ▷ the set of potential next states, each corresponds to a set of fluents
$\mathcal{P} : \mathcal{S}_{t+1} \rightarrow [0, 1]$ ▷ a probability distribution over the next states
Data : $\mathbb{A}$ ▷ action situation description
$s_t = \{f_1, f_2, ...\}$ ▷ set of facts that hold true at the current state
$\mu = \{\texttt{does}(ag_1, ac_1), \texttt{does}(ag_2, ac_2), ...\}$ ▷ joint action profile
Description : Get the post-transition state fluents and their probabilities that derive from performing some joint action in a pre-transition state.

Function GET-CONTROL-CONSEQS(*id, thres*)**:**
  $kv \leftarrow []$
  foreach instance of `?-query_rule(rule(`*id*`,`*control*`,`*pr*`,if Cond then Conseqs where Constr))` **do**
    if $n \leq thres$ **then** kv.APPEND(pr : `Conseqs`)
  $kv \leftarrow$ SORT-BY-DESCENDING-KEY(kv)
  $\mathcal{S}_{t+1} \leftarrow \{\{\}\}$
  $\mathcal{P}(\{\}) = 1$
  for $(pr$: *conseqs*$)$ pair in kv **do** // loop over activated control rules
    /* *conseqs* = [c_{11} and c_{12} and ... withProb p_1,
    c_{21} and c_{22} and ... withProb p_2,
    ...] */
    /* check that every fluent in the consequences is consistent with the facts already established in the potential next states */
    for $(c_{i1}$ and c_{i2} and ... withProb $p_i)$ in *conseqs* **do**
      /* c_{ij} refers to the j-th fluent of the i-th consequence in the list of consequences induced by the control rule */
      $\mathcal{C}_i \leftarrow \{c_{ij}\}_{c_i = c_{i1} \text{ and } c_{i2} \text{ and } ...}$
      for (c_{ij}, s_{t+1}) in $\mathcal{C}_i \times \mathcal{S}_{t+1}$ **do**
        if `?-incompatible(`c_{ij}, s_{t+1}`)` returns `true` **then** move to the next $(pr : conseqs)$ pair // aka go to line 8
    /* the activated rule consequences are consistent with $\mathcal{S}_{t+1}$ */
    $\mathcal{S}'_{t+1} \leftarrow \{\}$
    for $s_{t+1} \in \mathcal{S}_{t+1}$ **do**
      for $(c_{i1}$ and c_{i2} and ... withProb $p_i)$ in *conseqs* **do**
        $\mathcal{C}_i \leftarrow \{c_{ij}\}_{c_i = c_{i1} \text{ and } c_{i2} \text{ and } ...}$
        $\mathcal{S}'_{t+1} \leftarrow \mathcal{S}'_{t+1} \cup \{s_{t+1} \cup \mathcal{C}_i\}$
        $\mathcal{P}(s_{t+1} \cup \mathcal{C}_i) \leftarrow \mathcal{P}(s_{t+1}) \cdot p_i$
    $\mathcal{S}_{t+1} \leftarrow \mathcal{S}'_{t+1}$
  for $(f_i, s_{t+1}) \in s_t \times \mathcal{S}_{t+1}$ **do** // drag compatible facts from s_t over to s_{t+1}
    if `?-incompatible(`f_i, s_{t+1}`)` returns `false` **then** $s_{t+1} \leftarrow s_{t+1} \cup \{f_i\}$
  return $\mathcal{S}_{t+1}$, $\mathcal{P}$

Algorithm 3: Function BUILD-GAME-ROUND(*id*, *thres*).

Input : id ▷ string
$thres$ ▷ non-negative integer
Output : γ ▷ game round
F ▷ set of fluents assigned to γ's terminal nodes
$\tau : Z \to \{0, 1\}$ ▷ function mapping whether termination conditions are met at a terminal node ($\tau = 1$) or not ($\tau = 0$).
Data : $\mathbb{A}$ ▷ action situation description
$s_t = \{f_1, f_2, ...\}$ ▷ set of facts
Description : Given a state characterized by a set of facts, model all the ways by which it may evolve as a game round.

Function BUILD-GAME-ROUND(*id, thres*)**:**
  $k \leftarrow 1$, $X \leftarrow \{k\}$, $x_0 \leftarrow k$, $k++$
  $E \leftarrow \{\}$
  /* set players to be those participants that can take some action */
  $M \leftarrow$ GET-SIMPLE-CONSEQS(id, choice, $thres$) = $\{\texttt{can}(ag_1, ac_1), \texttt{can}(ag_2, ac_2), ...\}$
  $P \leftarrow \{ag_i \mid \exists \texttt{can}(ag_i, ac) \in M\}$
  /* STEP 1: Build the game tree in a breadth-first manner */
  $w \leftarrow \{x_0\}$, $w' \leftarrow \{\}$ // current and next information sets
  $W \leftarrow \{\}$, $\mathscr{A} \leftarrow \{\}$ // information partition and actions
  for $player \in P$ **do**
    $W_{player} \leftarrow \{w\}$, $W \leftarrow W \cup \{W_{player}\}$
    $A(w) \leftarrow \{ac \mid \texttt{can}(player, ac) \in M\}$, $\mathscr{A} \leftarrow \mathscr{A} \cup \{A(w)\}$
    for $x \in w$ **do**
      $T(x) \leftarrow player$
      for $action \in A(w)$ **do**
        $X \leftarrow X \cup \{k\}$, $w' \leftarrow w' \cup \{k\}$
        $E \leftarrow E \cup \{(x, k)\}$, $label(x, k) \leftarrow action$
        $k++$
    $w \leftarrow w'$, $w' \leftarrow \{\}$
  /* STEP 2: Get the facts and chance moves at the terminal nodes */
  $p \leftarrow \{\}$ // probability over chance moves
  for $z \in Z \subseteq X$ **do** // Z is the subset of terminal nodes
    // action profile from root to terminal node
    $\mu = \{\texttt{does}(ag, ac)\}_{\forall ag = T(x_i),\ ac = label(x_i, x_{i+1}) \mid (x_i, x_{i+1}) \in \text{PATH}(x_0, z)}$
    $\mathbb{A} \leftarrow \mathbb{A} \cup \mu$ // assert action profile into database
    if `?-terminal` returns `true` **then** $t \leftarrow 1$ **else** $t \leftarrow 0$
    $\mathcal{S}_{t+1}, \mathcal{P} =$ GET-CONTROL-CONSEQS(id, $thres$)
    if $\mathcal{S}_{t+1} = \{s_{t+1}\}$ **then** $F(z) = s_{t+1}$, $\tau(z) = t$ // no stochastic effects
    else // stochastic effects
      $T(z) \leftarrow chance$
      for $s^i_{t+1} \in \mathcal{S}_{t+1}$ **do**
        $X \leftarrow X \cup \{k\}$, $E \leftarrow E \cup \{(z, k)\}$
        $p_z(z, k) \leftarrow \mathcal{P}(s^i_{t+1})$
        $F(k) \leftarrow s^i_{t+1}$, $\tau(k) \leftarrow t$, $k++$
      $p \leftarrow p \cup \{p_z\}$
    $\mathbb{A} \leftarrow \mathbb{A} \setminus \mu$ // remove actions from the database
  $\gamma = (P, (X, E), T, W, \mathscr{A}, p)$
  return γ, F, τ

Algorithm 4: Function BUILD-FULL-GAME(id, $thres$, max).

Input : id ▷ string
$thres$ ▷ non-negative integer
max ▷ non-negative integer
Output : Γ ▷ extensive-form game
$\mathcal{F}$ ▷ set of fluents assigned to γ's state nodes
Data : $\mathbb{A}$ ▷ action situation description
Description : Given an action situation description, generate its extensive-form game semantics.

Function BUILD-GAME-ROUND(id, $thres$, max)**:**
  $\phi \leftarrow$ GET-SIMPLE-CONSEQS(id, boundary, $thres$) $= \{\texttt{participates}(ag_1), ...\}$
  $\mathbb{A} \leftarrow \mathbb{A} \cup \phi$
  $\rho \leftarrow$ GET-SIMPLE-CONSEQS(id, position, $thres$) $= \{\texttt{role}(ag_1, r_1), ...\}$
  $\mathbb{A} \leftarrow \mathbb{A} \cup \rho$
  $s_0 \leftarrow \{\}$
  foreach instantiation f_i of `?-initially(F)` **do** $s_0 \leftarrow s_0 \cup \{f_i\}$ `// initial facts`
  $P = \{ag_i\}_{\forall \texttt{participates}(ag_i) \in \phi}$
  $X \leftarrow \{1\}$, $x_0 \leftarrow 1$, $E \leftarrow \{\}$, $W \leftarrow \{\{\}, ..., \{\}\}_{\forall i \in P}$, $\mathscr{A} \leftarrow \{\}$, $p \leftarrow \{\}$
  $\mathcal{F}(1) \leftarrow s_0$
  $round(1) \leftarrow 0$
  $\mathcal{Q} \leftarrow$ QUEUE(1)
  while $\mathcal{Q}$ is not empty **do**
    $n \leftarrow \mathcal{Q}$.POP()
    if $round(n) \geq max$ **then continue**
    $s_t \leftarrow \mathcal{F}(n)$
    $\mathbb{A} \leftarrow \mathbb{A} \cup \{s_t\}$ `// assert node facts into database`
    if `?-terminal` returns `true` **then** $\mathbb{A} \leftarrow \mathbb{A} \setminus \{s_t\}$, **continue**
    $\gamma, F, \tau \leftarrow$ BUILD-GAME-ROUND(id, $thres$)
    $\mathbb{A} \leftarrow \mathbb{A} \setminus \{s_t\}$
    `/* append game round to overall game tree - superindex` γ `denotes the elements from the game round */`
    for $x \in X^\gamma$ **do** $x \leftarrow x + n - 1$ `// node re-labeling`
    $X \leftarrow X \cup X^\gamma$, $E \leftarrow E \cup E^\gamma$
    for $x \in X^\gamma \setminus Z^\gamma$ **do** $T(x) \leftarrow T^\gamma(x)$
    for $p \in P$ **do** $W_p \leftarrow W_p \cup W_p^\gamma$
    for $A(w) \in \mathscr{A}^\gamma$ **do** $\mathscr{A} \leftarrow \mathscr{A} \cup \{A(w)\}$
    forall $x \in X^\gamma \mid T(x) = chance$ **do** $p \leftarrow p \cup \{p_x^\gamma\}$
    for $z \in Z^\gamma$ **do** $\mathcal{F}(z) \leftarrow F(z)$
    forall $z \in Z^\gamma$ **do**
      $round(z) \leftarrow round(n) + 1$
      if $\tau(z) = 0$ **then** $\mathcal{Q}$.PUSH(z)
  $\Gamma = (P, (X, E), T, W, \mathscr{A}, p)$
  return Γ, $\mathcal{F}$

Appendix B. Example game semantics

B.1. Iterated Prisoner's Dilemma

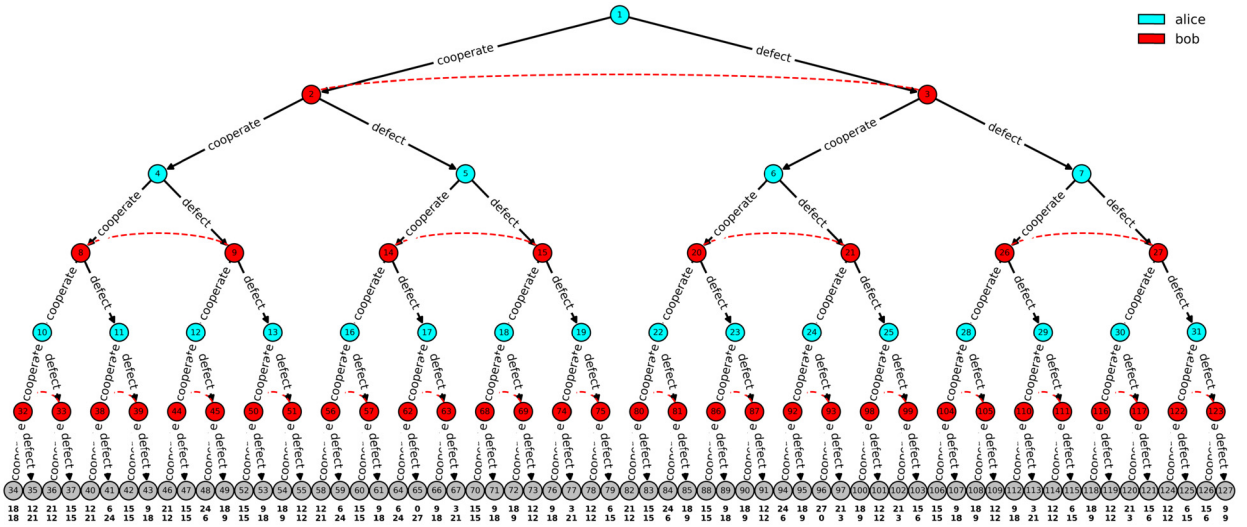


Fig. B.6. Game semantics for the iterated Prisoner's Dilemma with the default rule configuration.

Table B.4
Equilibrium strategies (top) and distribution over outcomes (bottom) for the default rule configuration of the iterated Prisoner's Dilemma game.

Agent	Information set	Action	Probability
alice	all	cooperate	0
		defect	1
bob	all	cooperate	0
		defect	1

Terminal node(s)	State fluents	Probability
127	`payoff(alice,9)`, `payoff(bob,9)`	1

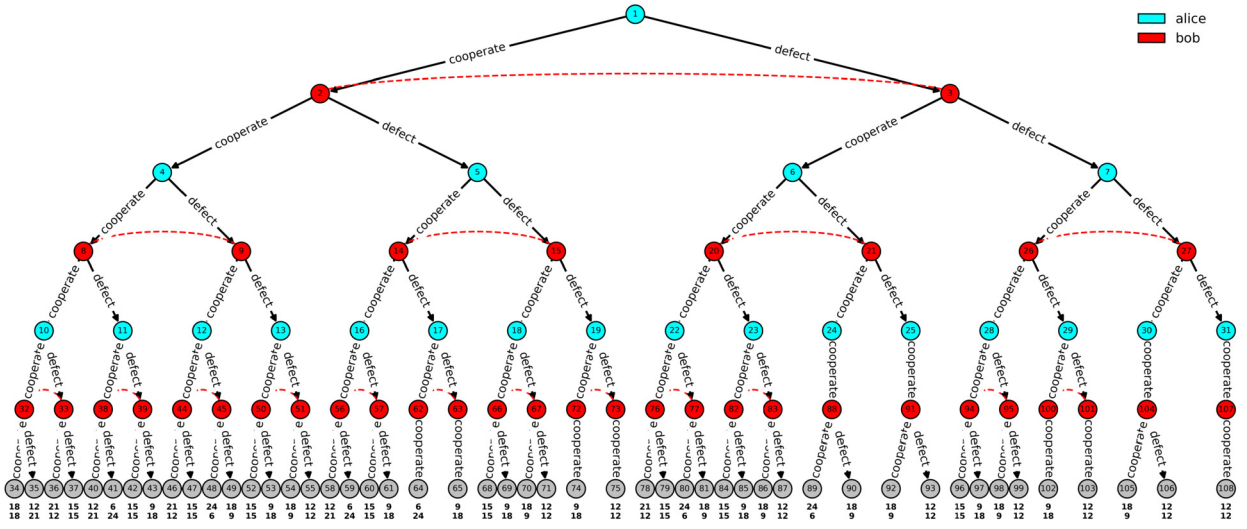


Fig. B.7. Game semantics for the iterated Prisoner's Dilemma with an additional rule for restricting to 2 the number of consecutive defections.

Table B.5
Equilibrium strategies (top) and distribution over outcomes (bottom) for the iterated Prisoner's Dilemma with an additional rule for restricting the number of consecutive defections. For the bottom table, only the state fluents that are common to all the outcomes with non-zero probability are shown.

Agent	Information set	Action	Probability
	{1}, {6}, {7}	cooperate	1/2
		defect	1/2
alice	{24}, {25}, {30}, {31}	cooperate	1
		defect	0
	all others	cooperate	0
		defect	1
	{2, 3}, {26, 27}, {14, 15}	cooperate	1/2
		defect	1/2
bob	{62, 63}, {72, 73}, {100, 101}, {107}	cooperate	1
		defect	0
	all others	cooperate	0
		defect	1

Terminal node(s)	State fluents	Probability
55		1
71, 75, 87, 93	`payoff(alice,12)`, `payoff(bob,12)`, ...	0.125
99, 103, 106, 108		0.0625

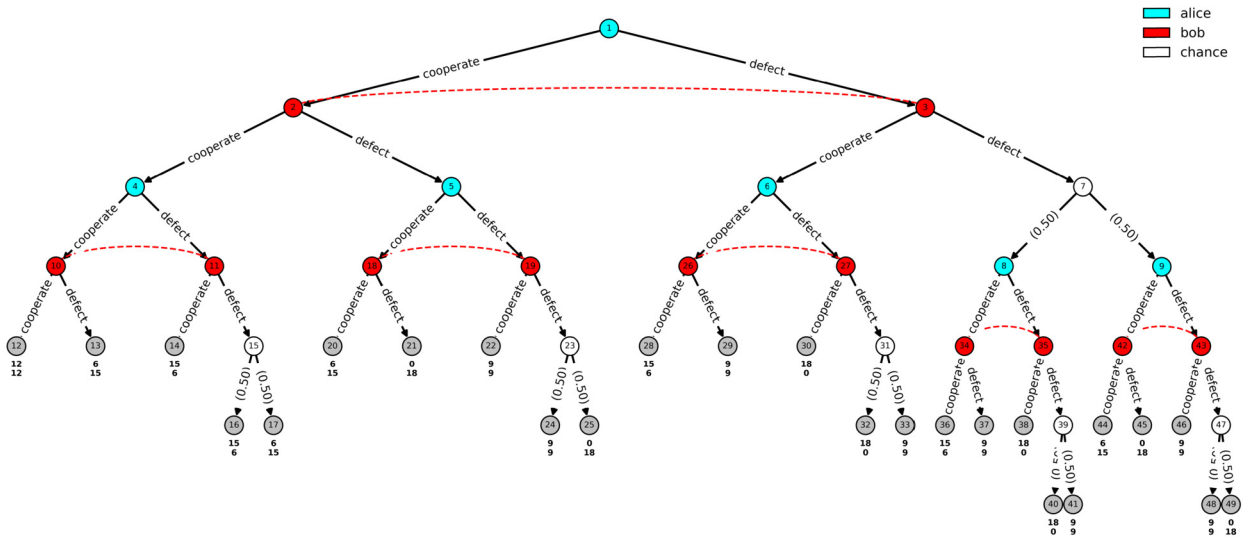


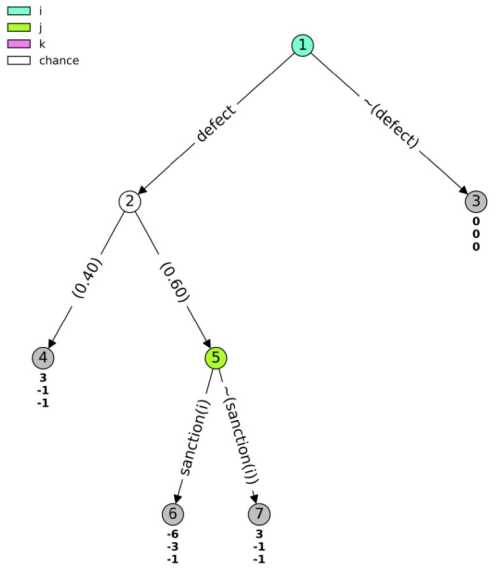
Fig. B.8. Game semantics for the iterated Prisoner's Dilemma with additional rules to change the outcome of a consecutive defection. For visualization purposes, termination conditions are met after agents play two game rounds instead of three.

Table B.6
Equilibrium strategies (top) and distribution over outcomes (bottom) for the iterated Prisoner's Dilemma with additional rules to change the outcome of a consecutive defection.

Agent	Information set	Action	Probability
alice	all	cooperate	0
		defect	1
bob	all	cooperate	0
		defect	1

Terminal node(s)	State fluents	Probability
40	`payoff(alice,18)`, `payoff(bob,0)`	0.25
41	`payoff(alice,9)`, `payoff(bob,9)`	0.25
48	`payoff(alice,9)`, `payoff(bob,9)`	0.25
49	`payoff(alice,0)`, `payoff(bob,18)`	0.25

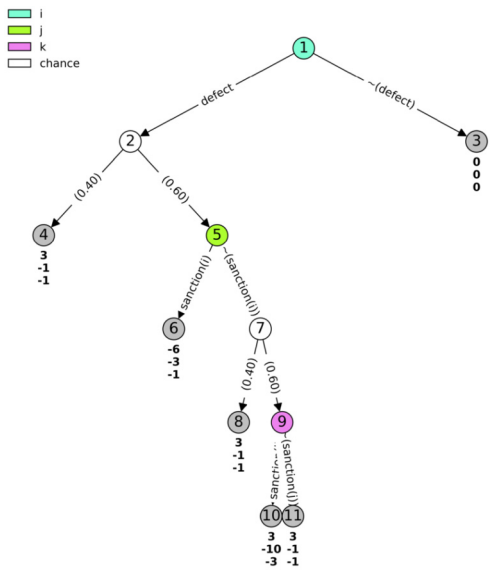
B.2. Axelrod's (meta)-norms game



(a)

	State fluents	p
1	payoff(i,0), payoff(j,0), payoff(k,0), time(0)	-
3	payoff(i,0), payoff(j,0), payoff(k,0), time(1)	0
4	payoff(i,3), payoff(j,-1), payoff(k,-1), time(1), (seen(j,i))	0.4
5	payoff(i,3), payoff(j,-1), payoff(k,-1), seen(j,i), time(1)	-
6	payoff(i,-6), payoff(j,-3), payoff(k,-1), time(2)	0
7	payoff(i,3), payoff(j,-1), payoff(k,-1), time(2)	0.6

(b)



(c)

	State fluents	p
1	payoff(i,0), payoff(j,0), payoff(k,0), time(0)	-
3	payoff(i,0), payoff(j,0), payoff(k,0), time(1)	0
4	payoff(i,3), payoff(j,-1), payoff(k,-1), time(1), (seen(j,i))	0.4
5	payoff(i,3), payoff(j,-1), payoff(k,-1), seen(j,i), time(1)	-
6	payoff(i,-6), payoff(j,-3), payoff(k,-1), time(2)	0
8	payoff(i,3), payoff(j,-1), payoff(k,-1), time(2), (seen(k,j))	0.24
9	payoff(i,3), payoff(j,-1), payoff(k,-1), seen(k,j), time(2)	-
10	payoff(i,3), payoff(j,-10), payoff(k,-3), time(3)	0
11	payoff(i,3), payoff(j,-1), payoff(k,-1), time(3)	0.36

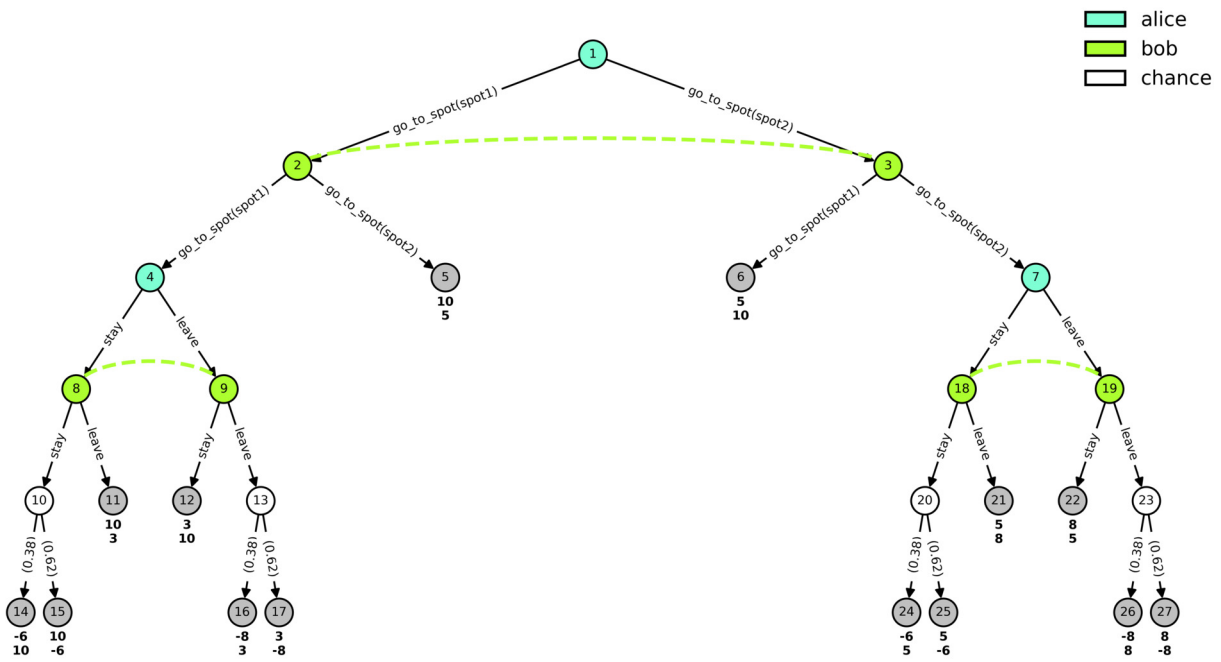
(d)

Fig. B.9. Semantics for Axelrod's norms (top) and metanorms (bottom) action situations, with the extensive game tree (left) and the corresponding state fluents (right). For the terminal nodes, their probability **p** induced by the equilibrium strategies and chance moves is also given.

Table B.7
Equilibrium strategies for Axelrod's norms and meta-norms games. For the norms game, only the first two rows apply (strategies for i and j). For the meta-norms game, all rows apply.

Agent	Information set	Action	Probability
i	{1}	defect ∼defect	1 0
j	{5}	sanction(i) ∼sanction(i)	0 1
k	{9}	sanction(j) ∼sanction(j)	0 1

B.3. Ostrom's fishing game



(a)

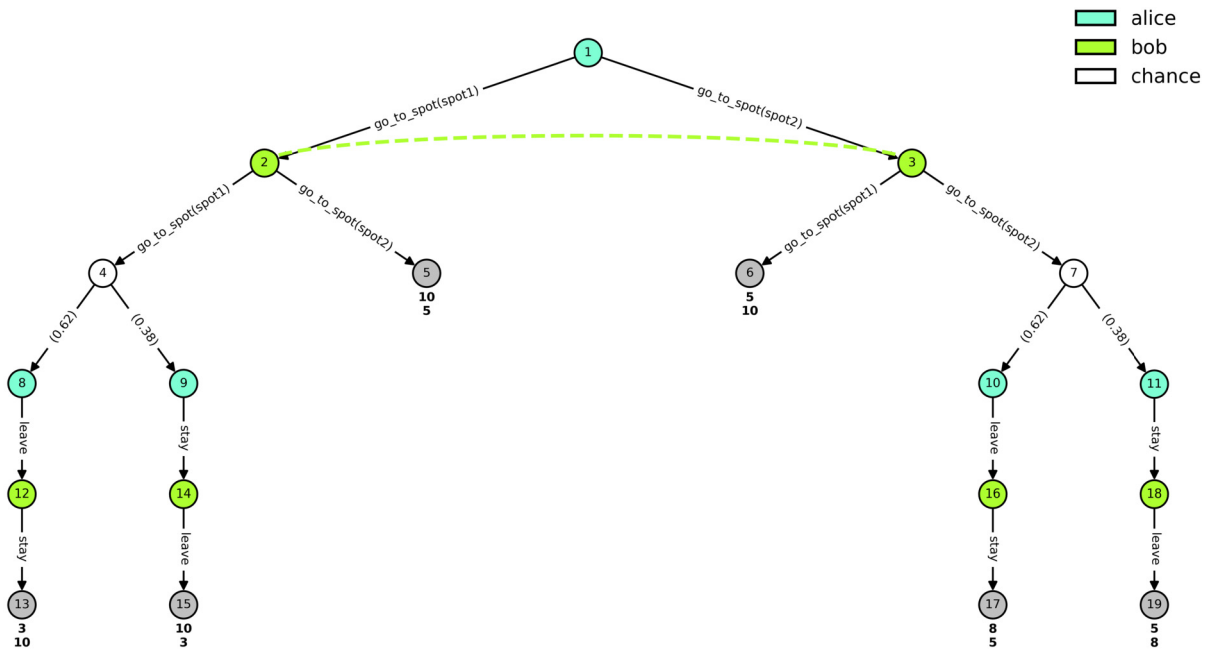
Agent	Information set	Action	Probability
alice	{1}	go_to_spot(spot1)	0.71
		go_to_spot(spot2)	0.29
	{4}	stay	0.82
		leave	0.18
	{7}	stay	0.41
		leave	0.59
bob	{2, 3}	go_to_spot(spot1)	0.84
		go_to_spot(spot2)	0.16
	{8, 9}	stay	1
		leave	0
	{18, 19}	stay	0.30
		leave	0.70

(b)

	State fluents	p		State fluents	p
1	at(alice, shore), at(bob, shore)	-	14	at(alice, spot1), at(bob, spot1), won_fight(bob)	0.18
4	at(alice, spot1), at(bob, spot1)	-	15	at(alice, spot1), at(bob, spot1), won_fight(alice)	0.31
5	at(alice, spot1), at(bob, spot2)	0.11	16	at(alice, spot2), at(bob, spot2), won_fight(bob)	0
6	at(alice, spot2), at(bob, spot1)	0.25	17	at(alice, spot2), at(bob, spot2), won_fight(alice)	0
7	at(alice, spot2), at(bob, spot2)	-	24	at(alice, spot2), at(bob, spot2), won_fight(bob)	0
11	at(alice, spot1), at(bob, spot2)	0	25	at(alice, spot2), at(bob, spot2), won_fight(alice)	0
12	at(alice, spot2), at(bob, spot1)	0.11	26	at(alice, spot1), at(bob, spot1), won_fight(bob)	0.01
21	at(alice, spot2), at(bob, spot1)	0.01	27	at(alice, spot1), at(bob, spot1), won_fight(alice)	0.01
22	at(alice, spot1), at(bob, spot2)	0.01			

(c)

Fig. B.10. Semantics for the fishers default action situation: game tree (a), equilibrium strategies (b) and state fluents with the probabilities over the terminal nodes induced by the equilibrium strategies and chance moves (c).



(a)

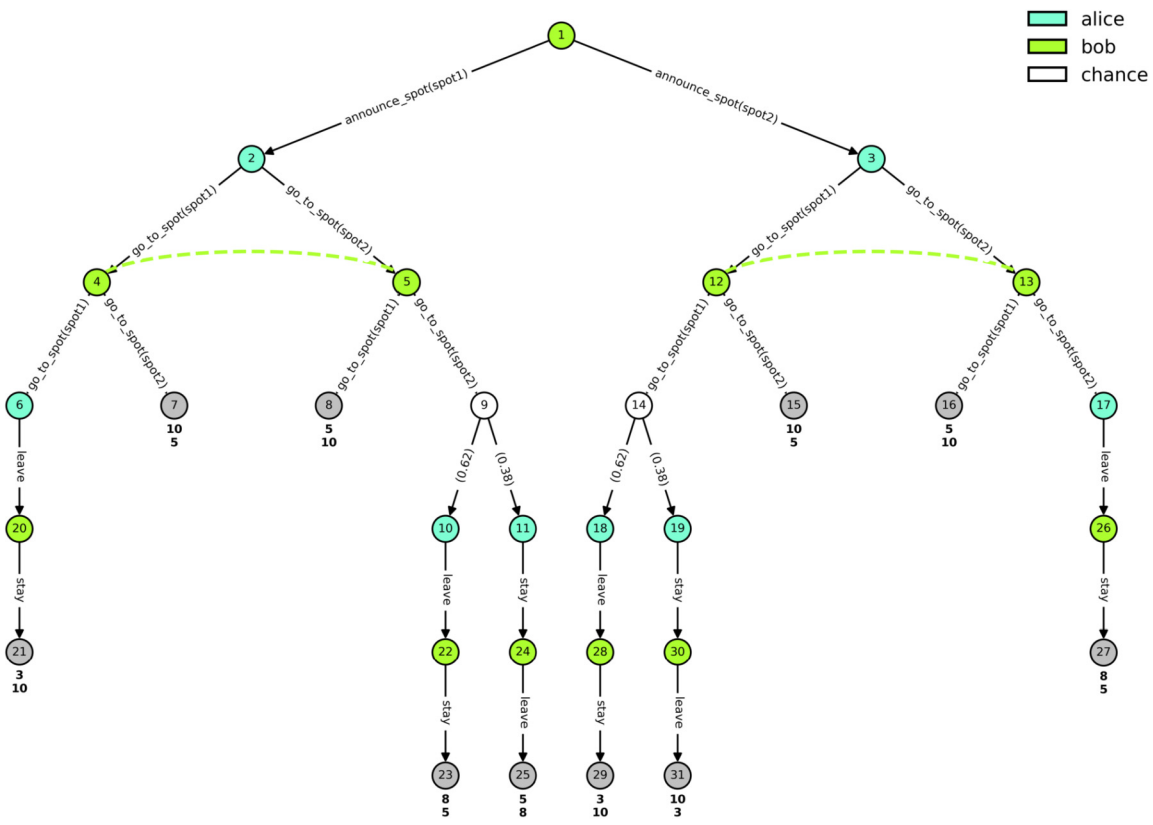
Agent	Information set	Action	Probability
alice	{1}	go_to_spot(spot1)	1
		go_to_spot(spot2)	0
	{8}	leave	1
	{9}	stay	1
bob	{2, 3}	go_to_spot(spot1)	1
		go_to_spot(spot2)	0
	{12}	stay	1
	{14}	leave	1

(b)

	State fluents	p
1	at(alice, shore), at(bob, shore)	-
5	at(alice, spot1), at(bob, spot2)	0
6	at(alice, spot2), at(bob, spot1)	0
8	at(alice, spot1), at(bob, spot1), won_race(bob)	-
9	at(alice, spot1), at(bob, spot1), won_race(alice)	-
10	at(alice, spot2), at(bob, spot2), won_race(bob)	-
11	at(alice, spot2), at(bob, spot2), won_race(alice)	-
13	at(alice, spot2), at(bob, spot1), won_race(bob)	0.62
15	at(alice, spot1), at(bob, spot2), won_race(alice)	0.38
17	at(alice, spot1), at(bob, spot2), won_race(bob)	0
19	at(alice, spot2), at(bob, spot1), won_race(alice)	0

(c)

Fig. B.11. Semantics for the fishers *first-in-time, first-in-right* action situation: game tree (a), equilibrium strategies (b) and state fluents with the probabilities over the terminal nodes induced by the equilibrium strategies and chance moves (c). The equilibrium strategies for information sets {10}, {11}, {16} and {18} have been omitted since they are not relevant for the game (i.e. agents never choose to go to spot 2).



(a)

Agent	Information set	Action	Probability
alice	{2}	go_to_spot(spot1) go_to_spot(spot2)	0 1
bob	{1}	announce_spot(spot1) announce_spot(spot2)	1 0
	{4, 5}	go_to_spot(spot1) go_to_spot(spot2)	1 0

(b)

	State fluents	*p*
1	at(alice, shore), at(bob, shore)	-
2	announced(bob, spot1), at(alice, shore), at(bob, shore)	-
3	announced(bob, spot2), at(alice, shore), at(bob, shore)	-
6	announced(bob, spot1), at(alice, spot1), at(bob, spot1), won_race(bob)	-
7	announced(bob, spot1), at(alice, spot1), at(bob, spot2)	0
8	announced(bob, spot1), at(alice, spot2), at(bob, spot1)	1.00
10	announced(bob, spot1), at(alice, spot2), at(bob, spot2), won_race(bob)	-
11	announced(bob, spot1), at(alice, spot2), at(bob, spot2), won_race(alice)	-
15	announced(bob, spot2), at(alice, spot1), at(bob, spot2)	0
16	announced(bob, spot2), at(alice, spot2), at(bob, spot1)	0
17	announced(bob, spot2), at(alice, spot2), at(bob, spot2), won_race(bob)	-
18	announced(bob, spot2), at(alice, spot1), at(bob, spot1), won_race(bob)	-
19	announced(bob, spot2), at(alice, spot1), at(bob, spot1), won_race(alice)	-
21	announced(bob, spot1), at(alice, spot2), at(bob, spot1), won_race(bob)	0
23	announced(bob, spot1), at(alice, spot1), at(bob, spot2), won_race(bob)	0
25	announced(bob, spot1), at(alice, spot2), at(bob, spot1), won_race(alice)	0
27	announced(bob, spot2), at(alice, spot1), at(bob, spot2), won_race(bob)	0
29	announced(bob, spot2), at(alice, spot2), at(bob, spot1), won_race(bob)	0
31	announced(bob, spot2), at(alice, spot1), at(bob, spot2), won_race(alice)	0

(c)

Fig. B.12. Semantics for the fishers *first-to-announce, first-in-right* action situation: game tree (a), equilibrium strategies (b) and state fluents with the probabilities induced over the terminal nodes by the equilibrium strategies and chance moves (c). Only the equilibrium strategies for the information sets that are actually visited during game play are included.

References

[1] E. Ostrom, Background on the Institutional Analysis and Development framework, Policy Stud. J. 39 (1) (2011) 7–27, https://doi.org/10.1111/j.1541-0072.2010.00394.x.
[2] E. Ostrom, Understanding Institutional Diversity, Princeton University Press, 2005.
[3] M. Black, Models and Metaphors: Studies in Language and Philosophy, Cornell University Press, Ithaca, NY, 1962.
[4] D. Cozort, J.M. Shields (Eds.), The Oxford Handbook of Buddhist Ethics, Oxford University Press, 2018.
[5] E. Ostrom, Governing the Commons, Cambridge University Press, 1990.
[6] J. Weymark, Social Welfare Functions, Oxford University Press, 2016, pp. 126–159, Ch. 5.
[7] L.L. Kiser, E. Ostrom, The Three Worlds of Action: A Metatheoretical Synthesis of Institutional Approaches, Michigan University Press, Ann Arbor, 1982, pp. 56–88, Ch. 2.
[8] S. Sarr, B. Hayes, D.A. DeCaro, Applying Ostrom's Institutional Analysis and Development framework, and design principles for co-production to pollution management in Louisville's Rubbertown, Kentucky, Land Use Policy 104 (2021) 105383, https://doi.org/10.1016/j.lusepol.2021.105383.
[9] T. Nguyen, T. Watanabe, Autonomous motivation for the successful implementation of waste management policy: an examination using an adapted Institutional Analysis and Development framework in Thua Thien Hue, Vietnam, Sustainability 12 (7) (2020) 2724, https://doi.org/10.3390/su12072724.
[10] D.N. Barton, K. Benavides, A. Chacon-Cascante, J.F. Le Coq, M.M. Quiros, I. Porras, E. Primmer, I. Ring, Payments for ecosystem services as a policy mix: demonstrating the Institutional Analysis and Development framework on conservation policy instruments, Environ. Policy Gov. 27 (5) (2017) 404–421, https://doi.org/10.1002/eet.1769.
[11] D.H. Cole, Laws, norms, and the Institutional Analysis and Development framework, J. Inst. Econ. 13 (4) (2017) 829–847, https://doi.org/10.1017/s1744137417000030.
[12] A. Mas-Colell, M.D. Whinston, J.R. Green, Microeconomic Theory, Oxford University Press, New York, 1995.
[13] Y. Shoham, M. Tennenholtz, On social laws for artificial agent societies: off-line design, Artif. Intell. 73 (1–2) (1995) 231–252, https://doi.org/10.1016/0004-3702(94)00007-n.
[14] S. Onn, M. Tennenholtz, Determination of social laws for multi-agent mobilization, Artif. Intell. 95 (1) (1997) 155–167, https://doi.org/10.1016/s0004-3702(97)00045-3.
[15] G. Andrighetto, G. Governatori, P. Noriega, L. van der Torre, Normative multi-agent systems (Dagstuhl seminar 12111), Dagstuhl Rep. 2 (3) (2012) 23–49, https://doi.org/10.4230/DagRep.2.3.23, http://drops.dagstuhl.de/opus/volltexte/2012/3535.
[16] C. Hahn, T. Phan, S. Feld, C. Roch, F. Ritz, A. Sedlmeier, T. Gabor, C. 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, pp. 234–241.
[17] P. Caillou, S. Aknine, S. Pinson, Searching Pareto optimal solutions for the problem of forming and restructuring coalitions in multi-agent systems, Group Decis. Negot. 19 (1) (2009) 7–37, https://doi.org/10.1007/s10726-009-9183-9.
[18] S.E.S. Crawford, E. Ostrom, A grammar of institutions, Am. Polit. Sci. Rev. 89 (3) (1995) 582–600, https://doi.org/10.2307/2082975.
[19] C. Frantz, M.K. Purvis, M. Nowostawski, B.T.R. Savarimuthu, nADICO: a nested grammar of institutions, in: Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 429–436.
[20] C.K. Frantz, S. Siddiki, Institutional Grammar 2.0: A Specification for Encoding and Analyzing Institutional Design, Public Administration, 2021.
[21] A. Ghorbani, G. Bravo, Managing the commons: a simple model of the emergence of institutions through collective action, Int. J. Commons 10 (1) (2016) 200–219, https://doi.org/10.18352/ijc.606.
[22] A. Smajgl, L.R. Izquierdo, M. Huigne, Modeling endogenous rule changes in an institutional context: the adico sequence, Adv. Complex Syst. 11 (02) (2008) 199–215, https://doi.org/10.1142/s021952590800157x.
[23] A. Ghorbani, P. Bots, V. Dignum, G. Dijkema, MAIA: a framework for developing agent-based social simulations, J. Artif. Soc. Soc. Simul. 16 (2) (2013), https://doi.org/10.18564/jasss.2166.
[24] M. Genesereth, N. Love, B. Pell, General game playing: overview of the AAAI competition, AI Mag. 26 (2005) 62–72, https://doi.org/10.1609/aimag.v26i2.1813.
[25] S. Schiffel, M. Thielscher, Representing and reasoning about the rules of general games with imperfect information, J. Artif. Intell. Res. 49 (2014) 171–206, https://doi.org/10.1613/jair.4115.
[26] M. Thielscher, GDL-III: a description language for epistemic general game playing, in: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, International Joint Conferences on Artificial Intelligence Organization, 2017, pp. 1276–1282.
[27] D. de Jonge, T. Trescak, C. Sierra, S. Simoff, R.L. de Mántaras, Using game description language for mediated dispute resolution, AI Soc. 34 (4) (2017) 767–784, https://doi.org/10.1007/s00146-017-0790-8.
[28] D. de Jonge, D. Zhang, GDL as a unifying domain description language for declarative automated negotiation, Auton. Agents Multi-Agent Syst. 35 (1) (2021), https://doi.org/10.1007/s10458-020-09491-6.
[29] R.B. Scherl, H.J. Levesque, Knowledge, action, and the frame problem, Artif. Intell. 144 (1–2) (2003) 1–39, https://doi.org/10.1016/s0004-3702(02)00365-x.
[30] D. Koller, A. Pfeffer, Representations and solutions for game-theoretic problems, Artif. Intell. 94 (1–2) (1997) 167–215, https://doi.org/10.1016/s0004-3702(97)00023-4.
[31] G.H. von Wright, Deontic logic, Mind 60 (237) (1951) 1–15, http://www.jstor.org/stable/2251395.
[32] M. Belzer, Deontic logic, in: Routledge Encyclopedia of Philosophy, Routledge, 1998.
[33] J. Morales, M. Lopez-Sanchez, J.A. Rodriguez-Aguilar, M. Wooldridge, W. Vasconcelos, Automated synthesis of normative systems, in: Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems, AAMAS '13, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2013, pp. 483–490.
[34] M.S. Fagundes, S. Ossowski, J. Cerquides, P. Noriega, Design and evaluation of norm-aware agents based on normative Markov decision processes, Int. J. Approx. Reason. 78 (2016) 33–61, https://doi.org/10.1016/j.ijar.2016.06.005.
[35] J. Szabo, J.M. Such, N. Criado, Understanding the role of values and norms in practical reasoning, in: N. Bassiliades, G. Chalkiadakis, D. de Jonge (Eds.), Multi-Agent Systems and Agreement Technologies, Springer International Publishing, Cham, 2020, pp. 431–439.
[36] D. Grossi, D. Gabbay, L. van der Torre, The norm implementation problem in normative multi-agent systems, in: Specification and Verification of Multi-agent Systems, Springer US, 2010, pp. 195–224.
[37] F. Lin, Situation Calculus, Foundations of Artificial Intelligence, vol. 3, Elsevier, 2008, pp. 649–669, Ch. 16.
[38] J. González-Díaz, I. García-Jurado, M.G. Fiestras-Janeiro, An Introductory Course on Mathematical Game Theory, American Mathematical Society and Real Sociedad Matemática Española, Providence, Rhode Island, USA and Madrid, 2010.
[39] S. Fatima, S. Kraus, M. Wooldridge, Principles of Automated Negotiation, Cambridge University Press, 2009.
[40] N. Gronewold, Game theory: climate talks destined to fail (Dec 2010), https://www.scientificamerican.com/article/game-theorist-predicts-failure-at-climate-talks/.
[41] H.W. Kuhn, 11. Extensive games and the problem of information, in: Contributions to the Theory of Games (AM-28), Vol. II, Princeton University Press, 1953, pp. 193–216.

[42] J.F. Nash, Equilibrium points in n-person games, Proc. Natl. Acad. Sci. USA 36 (1) (1950) 48–49, http://www.jstor.org/stable/88031.
[43] R.J. Aumann, Subjectivity and correlation in randomized strategies, J. Math. Econ. 1 (1) (1974) 67–96, https://doi.org/10.1016/0304-4068(74)90037-8.
[44] Y. Shoham, K. Leyton-Brown, Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations, Cambridge University Press, 2014.
[45] L. Hammond, J. Fox, T. Everitt, A. Abate, M. Wooldridge, Equilibrium refinements for multi-agent influence diagrams: theory and practice, in: Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '21, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2021, pp. 574–582.
[46] R. Axelrod, An evolutionary approach to norms, Am. Polit. Sci. Rev. 80 (04) (1986) 1095–1111, https://doi.org/10.2307/1960858.
[47] E. Ostrom, R. Gardner, J. Walker, Rules, Games, and Common-Pool Resources, University of Michigan Press, 1994.