December 2019

Towards a Computational Theory of Action, Causation and Power for Normative Reasoning

Giovanni SILENO [a,1], Alexander BOER [b] and Tom VAN ENGERS [b,a]
[a] *Informatics Institute, University of Amsterdam, Netherlands*
[b] *Leibniz Institute, University of Amsterdam/TNO, Netherlands*

Abstract. In order to effectively implement guidance structures in a computational social system, directives which are specified in general terms of duties and rights need to be transformed in terms of powers and liabilities attributed to social parties. The present paper is a work in progress report on an axiomatization of power structures in a logic programming setting, covering the intentional level in specifying actions, the connection between productive characterization of actions and causation, the default nature of action specifications, failures and omissions, the relations of causation and power, and the concept of interfering actions.

Keywords. Models of action, Causation, Power, Ability, Susceptibility, Event Calculus, Reactive rules, Normative Reasoning, Logic Programming, ASP

1. Introduction

For enabling automated normative reasoning, norms need to be represented in a computationally processable way, just as the world on which such norms are deemed to apply. Ideally, two types of normative reasoning can be distinguished: (a) *reasoning with norms*, i.e. applying given norms to qualify behaviour and situations (possibly to take decisions upon); (b) *reasoning about norms*, that can be further inflected in internal view (i.e. check whether a certain norm is *valid* and *applicable* with respect to a given set of norms) and external views to legal systems (i.e. whether the norm is *effective* in guiding behaviour and/or it is *efficient* in terms of costs required for its maintenance). Although the internal/external distinction is mostly evident in (b), a more attentive analysis shows that elements of the second re-enter in (a). More specifically, in order to be applied on a social system, i.e. to effectively implement guidance structures (for instance in a computational social system) *directives which are specified in general terms of duties and rights needs to be reinterpreted in terms of powers and liabilities attributed to social parties.*

There is a long-standing debate between proponents of a purely deontic approach to norms (in legal philosophy see e.g. MacCormick and Raz [1]: "Though powers are essential to the explanation of rights, they are not in themselves rights"), and "paritary" approaches to deontic and potestative categories as the one advanced by Hohfeld [2]. This debate is mirrored in logic and related computational disciplines, although most solutions starts implicitly or explicitly from some flavour of *deontic logic* (see e.g. [3]). Inspired by Hohfeld, a much smaller number of studies focuses on directed obligations and rights

[1] Corresponding Author: g.sileno@uva.nl. This research was partly funded by NWO (VWData project).

including powers (e.g. [4,5,6]). Relevant to this group, although different in spirit, is the contribution by [7], highlighting the *teleological* aspects of normative relations.

The research direction motivating this paper takes a somehow even more radical stance: even acknowledging the primary role of deontic categories to specify optimality (and sub-optimality) in the world, we recognize the crucial role of potestative categories to deal with the implementation of normative mechanisms in the social system, in the attempt to guide it towards this optimality. Indeed, at the level of social system, *everything boils down to power structures, enabling or disabling action* (institutional and physical). Accepting this formulation, it is crucial to be able to model and reason with power, as well as with related concepts, as action and causation. The present paper can be seen as a work in progress report on an axiomatization in a *logic programming* (LP) setting.[2]

2. Representing action

Procedural, productive and intentional characterizations It is generally acknowledged that three general characterizations of actions exist in human language, mapping to three levels of abstraction [8]: **task** (e.g. "Brutus stabbed Ceasar"), **outcome** ("Brutus killed Ceasar"), and **intent** ("Brutus murdered Ceasar").[3] More in detail:[4]

- The *behavioural* or *procedural* characterization relates to the task abstraction level, describing the type of behaviour that the agent has just followed (or is following, if the action is not atomic). We will denote it as `performs(X, A)`, as e.g. in `performs(brutus, stabbing)`.
- The *productive* characterization relates to the outcome level, describing the result that the agent's behaviour has produced. We will denote it as `brings(X, R)` as e.g. in `brings(brutus, dead)`.
- The *purposive* or *intentional* characterization is associated with the intent level, describing the intent which drives the agent's behaviour, whose content can be either procedural `aims(X, A)` as e.g. `aims(brutus, stabbing)` or productive `aims(X, R)`, as e.g. in `aims(brutus, dead)`.

Definition of actions Introducing a general predicate `does` for actions, we can rewrite the variations of the initial example in terms of characterizations:

```
does(brutus, stabbing) <-> performs(brutus, stabbing).
does(brutus, killing) <*> brings(brutus, dead).
does(brutus, murdering) <->  aims(brutus, killing), does(brutus, killing)
```

where `<->` indicates logical equivalence, and `<*>` stands for a *default* inference mechanism that will be investigated further in the text. At second sight, we note that the task-form (and similarly the `does` predicate) denotes the performance of an attempt, but in no case it implies that the associated result has been achieved. In general, result is defined as *completion* of the action (in the sense of successful execution), as e.g. `stabbed` or `killed`, and, in case of actions specified with a productive characterization, as an *effect*

[2] A prototype implementation in *answer set programming* (ASP) of most ideas presented here is publicly available at `http://leibnizcenter.org/resources/JURIX2019/actions.lp`.

[3] On similar lines, Sartor [7] considers the procedural and productive characterisations for the types of actions to be used for norm modelling. Amongst others, Clark and Clark [9] include also a *stative* characterisation.

[4] As a convention, we use the *-ing* verbal form for identifiers denoting the action as a *process* or performance, and the *-ed* form for the action denoted as an *object*, or act.

in the world, as e.g. `dead`. Note that the outcome-form specifies the final result, but does not necessarily refer to intent (as in case of accidents). The intent-form makes instead clear that the outcome of the action is performed with intent.

General properties Procedural characterizations can be associated to immediate intents (cf. Searle's *intention-in-action*, or by seeing intentions as selected plans, as in Bratman's account, basis for most BDI agent architectures):

```
performs(brutus, stabbing) -> aims(brutus, stabbing).
```

Intentional procedural content can be brought to productive:

```
aims(brutus, stabbing) <-> aims(brutus, stabbed).
```

By definition, all actions comes along with an implicit productive content, e.g.:

```
performs(brutus, stabbing) <*> brings(brutus, stabbed).
```

The symbol `<*>` was used above to highlight that the logical equivalence between performance and outcome does not always hold, as performances cover also failed attempts of action. We propose here a possible logical model, in the simplifying assumption of dealing only with atomic actions (i.e. their duration is irrelevant w.r.t. the model granularity). Clearly, if an act has been completed, then performance has occurred:

```
brings(brutus, stabbed) -> performs(brutus, stabbing).
```

In contrast, we can assume that performance is completed *by default*, unless it is known otherwise. We introduce then a *strong negation* predicate `neg`, but we also rely on the unary operator *default negation* `not` provided by the logic programming semantics:

```
performs(brutus, stabbing), not neg(brings(brutus, stabbed))
  -> brings(brutus, stabbed).
```

Note that, because actions of any characterization can be described in the task form, this property is inherited by the `does` predicate. In sum, by generalizing these examples we can identify a few axioms mapping *observations of performances* (`performs/2`), *attribution of causal responsibilities* (`brings/2`) and of *intentions* (`aims/2`), to and from *action descriptions* (`does/2`).

Perfect, imperfect actions, etc. Let us consider actions identified by a task description `A` and an outcome description `R`, related by the predicate `actionResult/2`. Let us consider that performance has a certain duration, but that the production of the outcome is (qualitatively) immediate. The following qualifications of an action `A` can be defined as conjunctions of `does(X, A)` and `actionResult(A, R)` with these other conditions:

- *perfect action*: `brings(X, R)`
- *imperfect action*: `neg(brings(X, R))`
- *ongoing action*: `not(brings(X, R))`
- *successful intention*: `aims(X, R), brings(X, R)`
- *failed intention*: `aims(X, R), neg(brings(X, R))`
- *ongoing attempt*: `aims(X, A), not(brings(X, R))`

where the `not/1` predicate is true if no conclusion about the term is possible, i.e. `not(P)` is true when `not P` and `not neg(P)` are true. So, by relying on the idea of imperfection, action can be defined *negatively*:

```
does(X, neg(A)) <-> imperfect(does(X, A)).
```

meaning that the action has been performed, but has not reached the expected result (*failure*). Note in contrast how `neg(does(X, A))` means that performance has not been initiated (*omission*).

3. Representing causation

In a computational system, causal mechanisms triggered by an action `A` performed by an agent `X` in condition `C` and resulting in producing or consuming an object `r`, can be implemented as *reactive rules*, similarly to *event-condition-action* (ECA) production systems:

```
performs(X, A) : holds(C) => +r.  % initiation
performs(X, A) : holds(C) => -r.  % termination
```

In our case, consequences (neglecting temporal aspects) consist in the initiation (`+`) or the termination (`-`) of one or more objects.

Causation in logical reasoning At further inspection, events as e.g. `performs(X, A)` have an implicit temporal annotation, because the agent might perform several times the same type of action. Thus, assuming actions to be atomic (immediate) and interleaved (an actor cannot performs the same action twice at the same moment), `performs(X, A, T)` would denote a well-specified action instance. Further, in the moment in which we are dealing with time, dynamic facts have to be transformed into fluents: any (predicate) object `O` requires to be situated in time, as in `holds(O, T)`. Neglecting the enabling condition `C`, causal mechanisms could be then rewritten by making explicit the *change* of state for the fluent caused by the action:

```
performs(X, A, T), initiates(A, R), neg(holds(R, T-1)) -> holds(R, T).
performs(X, A, T), terminates(A, R), holds(R, T-1) -> neg(holds(R, T)).
```

(note that that, written in this form, `A` is an action type, while `R` is an object instance.) Unfortunately, these axioms are not sufficient for a logically sound reasoning. As shown in *situation calculus* [10], *event calculus* [11] and functionally similar solutions, additional axioms are required to capture *inertia*, *circumscription* and related epistemic properties. Let us consider for instance the simplest version of event calculus:

```
%% event calculus axioms (F fluent, A action type, T, T1, T2 times)
holds(F, T) :- initially(F), not clipped(0, F, T).
holds(F, T2) :- occurs(A, T1), initiates(A, F, T1), T1 < T2,
                not clipped(T1, F, T2).
clipped(T1, F, T2) :- occurs(E, T), T1 <= T,  T < T2, terminates(A, F, T).
```

Here, actions and fluents are reified as terms rather than as predicates. Intuitively, this is because change occurs at a meta-level with respect to the level of objects, and then everything has to be brought at meta-level to reason with it. In contrast, the notation of *reactive rules* enables in principle to abstract temporal attributes, as it introduces constraints only at the level of events. The following reactive rule implements a causal mechanism:

```
performs(X, A): initiates(A, R) => +R.
```

but it corresponds to a logical dependence at event level (`+` act as a unary predicate instead of an operator). Then, some other computational mechanism is responsible for executing the initiation and termination of fluents. For their compactness, it is tempting to maintain the description of causal mechanisms as reactive rules separated from that of necessary constraints holding between the objects, even knowing that they are not independent: certain causal mechanisms can create implicit constraints, as well as given constraints can inhibit certain causal ramifications. However, it is important to remind here that it is possible to semantically unify them, e.g. by using event calculus.

4. Representing power

Power—of an agent `X` towards an object `Y` to obtain a consequence `R` (concerning `Y`) by performing an action `A`—can be seen as the reification of a causal mechanism:

```
power(X, Y, A, R) <-> [performs(X, A) => +R(Y)].
```

The biconditional can be nested in the reactive rule:

```
performs(X, A) : power(X, Y, A, R) => +R(Y).
```

unveiling that the `initiates` predicate seen above is nothing else than a coarser description of `power`. With respects to conditions, power, even more when acting on symbolic objects (as for institutional power), is grounded on three qualification processes: (1) parties `X` and `Y` qualify to certain roles; (2) action `A` qualifies to a certain type/form; (3) context (here implicit, typically concerning where and when and the absence of overruling by another normative source). Each of these components brings conditions on the application of the causal mechanism:

```
power(X, Y, A, R) :- role(X, x), role(Y, y),
  action(A, a), actionResult(A, R), context(C, c).
```

Ability and susceptibility In the general causal interpretation, power primarily addresses the agent party (the one performing the action), so it can be renamed as *ability*:

```
ability(X, Y, A, R) <-> power(X, Y, A, R).
```

In duality, we can define the notion of *susceptibility* by primarily addressing the recipient party. A recipient is susceptible to an action (and then to the agent performing it) if it suffers a change because of its occurrence:

```
susceptibility(Y, X, A, R) <-> power(X, Y, A, R).
```

Negative powers By analogy to physics, in which forces can be attractive and repulsive, given a certain power, we can define its opposite by changing the sign of the outcome (cf. negative power/liability in [12]):

```
neg-power(X, Y, A, R) <-> power(X, Y, A, neg(R))
```

On the other hand, we can define the absence of power as the irrelevance of the action with respect to a certain outcome:

```
no-power(X, Y, A, R) <-> not power(X, Y, A, R), not neg-power(X, Y, A, R).
```

Negative susceptibilities and no-susceptibilities can be defined accordingly.

Preparatory/interfering actions, enabling/disabling powers An action `IA` interferes with an action `A` if, when the first is performed, it inhibits the outcome usually expected for performing the second. This notion is crucial for defining e.g. protection measures against interference as for *freedom of speech* (see e.g. [7]). Interestingly, it can be expressed in terms of powers; as a matter of facts, the interfering action modifies the power associated to the action target of the interference. The modification can be *structural* (it holds after `IA`'s completion) or *contingent* (it holds as long as the performance of `IA` is occurring), constraints that can be captured respectively at event level and at object level:

```
% structural (at event level)
power(Z, power(X, Y, A, R), IA, neg)
  <-> [ performs(Z, IA) => +neg(power(X, Y, A, R)). ]
```

```
% contingent (at object level, neglecting the time variable T)
power(Z, power(X, Y, A, R), IA, neg)
  <-> [ not performs(Z, IA) -> power(X, Y, A, R).
        performs(Z, IA) -> neg(power(X, Y, A, R)). ]
```

Enabling powers, associated for instance to *preparatory* or *support actions*, can be described in a dual way.

5. Conclusions and future developments

Implicitly or explicitly, most systems referring to regulations, policies and similar constructs in the computational domain refer to some form of *deontic logic*. Plausibly because of the strict control structure inherent to computational systems, the potestative category is usually neglected. However, because computational systems are becoming more and more social systems with *de facto* decentralized control structures, it becomes crucial to form a theory of power, so that institutional design in computational settings can intervene directly at the *social coordination* level of the guidance problem. In principle, this representational standpoint should help to study the entrenchments holding between physical and institutional actions.

Directed by this higher-order goal, the present paper presents our starting point for an operational axiomatization of power structures in a logic programming setting, motivated by recent results in LP research and applications. It explicitly introduces the intentional level in specifying actions, it elaborates on the connection between productive characterization of actions and causation, it defines a way to compute failures and omissions, and establishes a connection between causation, ability/susceptibility and power, enabling a definition of interfering actions. Future extensions of this work will focus on a wider number of institutional patterns (ex-ante vs ex-post enforcement, punishment-based vs reward-based enforcement, delegation, etc.) and concepts (recklessness, negligence, etc.).

References

[1] N. MacCormick and J. Raz. Voluntary Obligations and Normative Powers. *Proceedings of the Aristotelian Society*, 46(1972):59–102, 1972.

[2] W. N. Hohfeld. Fundamental Legal Conceptions as Applied in Judicial Reasoning. *The Yale Law Journal*, 26(8):710–770, 1917.

[3] D. M. Gabbay, J. Horty, and X. Parent, editors. *Handbook of Deontic Logic and Normative Systems*. College Publications, 2013.

[4] L. Lindahl. *Position and Change: A Study in Law and Logic*. Synthese Library. Springer, 1977.

[5] D. Makinson. On the formal representation of rights relations. *Journal of Philosophical Logic*, 15, 1986.

[6] A.J.I. Jones and M. Sergot. A Formal Characterisation of Institutionalised Power. *Journal of IGPL*, 1996.

[7] G. Sartor. Fundamental Legal Concepts: A Formal and Teleological Characterisation. *Artificial Intelligence and Law*, 14(1):101–142, 2006.

[8] John F. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. MIT Press, 2000.

[9] H. H. Clark and E. V. Clark. *Psychology and Language: An Introduction to Psycholinguistics*. Harcourt Brace Jovanovich, 1977.

[10] R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.

[11] M. Shanahan. The Event Calculus Explained. *Artificial Intelligence Today*, pages 409–430, 1999.

[12] G. Sileno, A. Boer, and T. van Engers. On the Interactional Meaning of Fundamental Legal Concepts. In *Proceedings of JURIX 2014*, volume FAIA 271, pages 39–48, 2014.