ACE modular framework for computational ethics : dealing with multiple actions, concurrency and omission

Gauvain Bourgne, Camilo Sarmiento, Jean-Gabriel Ganascia

ACE modular framework for computational ethics : dealing with multiple actions, concurrency and omission

Gauvain Bourgne[1] , **Camilo Sarmiento**[1] , **Jean-Gabriel Ganascia**[1]
[1]Sorbonne Université, CNRS, LIP6, 75005 Paris, France
{gauvain.bourgne, camilo.sarmiento, jean-gabriel.ganascia}@lip6.fr

Abstract

This paper presents the ACE modular framework (Action-Causality-Ethics), a modular and declarative logic-based framework for representing and applying multiple ethical principles. We clearly separate modeling concerns about dynamics and ethics, using reification to explicitly represent generic meta-rules to be contrasted with specific domain knowledge. This allows us to formalize ethical principles as methods for ethical assessment of actions and plans and specify what additional domain information are needed for each of them in addition to the factual description of the unfolding of events. The architecture presented here is based on an action model allowing concurrency and multiple agents and we discuss how this affects the ethical assessment process and allows precise modelling of omission.

1 Introduction

With the increased ubiquity of Artificial Intelligence, there has been a growing demand for trustworthy AI, offering guarantees that decisions made by artificial agents are transparent, explainable and ethical. Computational ethics has emerged as a subfield of Artifical Intelligence, concerned with ensuring ethical behaviours in AI applications. As for humans, the very question of what constitutes an ethical behaviour is the topic of many arguments, forming the core of normative ethics, different schools have given rise to different ethical principles, with a classic divide between consequentialistic approaches and deontological ones.

A crucial concern in computational ethics is thus to put forth explicit representation of these ethical principles. To represent them in a unified way, we need some common representation and an architecture that can assess actions or plans according to multiple principles. We focus here on explicit formalization of ethical principles using logic and allowing practical implementation.

A number of contributions have been proposed to model ethical reasoning (see Tolmeijer et al. (2021) for a recent survey), but they are often focused on a single principle. Some noteworthy steps in the direction of a general architecture are the HERA project (Lindner, Bentzen, and Nebel, 2017) and the modular ASP framework proposed by Berreby, Bourgne, and Ganascia (2017). The first one is based on structural equations model abstracting away the whole dynamics in a causal model, while the second one proposes a modular logic program in ASP based on an adaptation of event calculus. We use this one as a basis for our proposal as it is more declarative and modular and based on richer models of actions which are used in planning and commonsense reasoning.

Thus, building upon Berreby, Bourgne, and Ganascia (2017), we present here the ACE modular framework (Action-Causality-Ethics), which divide the overall reasoning model in three layers : (i) an Action model, which is a devoid of ethical consideration and focus on modeling factually the dynamics of the situation, (ii) a Causality analysis layer, which is an intermediate analysis layer to enrich the concrete event trace with more abstract concepts, focused here on determining causal relation between events, (iii) the core Ethical layer, which does the actual ethical assessment of the different actions or scenarios that are provided by the former layer. This approach takes great care to separate different modeling concerns. First, using this three-layers architecture, we clearly separate concerns related to the modeling of the dynamics of the situation from purely ethical concerns (with causality acting as a bridge). Second, at each level, we strive to separate general mechanisms from applications specific inputs. This distinction is emphasized by a reification approach where all domain specific knowledge are expressed (after grounding) as a set of grounded facts, while general axiom takes the form of meta rules upon this reified knowledge. This modeling principle allows us to put forth explicit logical representation of ethical principles as each general set of axioms of the ethical layer (called Theories of the Right) defines admissibility for a given principle.

While our proposal adopts the modular architecture of Berreby, Bourgne, and Ganascia (2017), it is more general as it can deal with concurrent actions, allow representation of multiple agents and actions and propose an explicit representation of omitted actions, which differs from that of Berreby, Bourgne, and Ganasia (2018) in that it handles concurrency. Indeed, by

allowing concurrency, we can further distinguish between omissions that are forced by other actions (not doing something because I chose to do something else) and those that are really a choice not to do something.

The structure of this article follows the architecture of the ACE framework, with Section 2 discussing the action model, Section 3 presenting the causal analysis and Section 4 describing the ethical layer. Section 5 then concludes.

2 Action model

The action model is the foundation of our framework as it indicates how to express the dynamics of our system, specifying how the different actions or events can affect the world depending on their execution context.

Drawing inspiration from PDDL and discrete event calculus (Mueller, 2008) (DEC), we present here a basic model, which can be seen as a reified version of a STRIPS-fragment of DEC.

In our adaptation, indirect consequences are handled by "automatic event", which are akin to exogenous events in PDDL or to trigger axioms in DEC. These events occur whenever all their preconditions hold and can in turn trigger other ones, allowing reasoning over chains of events. We consider here a deterministic setting, meaning that both the effect of actions and the triggering of automatic events can be uniquely determined: it is thus always possible to infer without ambiguity the state in which one or more events will lead us to or which automatic event(s) will occur in a given state.

2.1 Action context

The *action context* consists of an *event specification* which is domain dependent and of an *initial situation*. The *event specification* defines the existing events, their type (action or automatic event, expressed with domain predicates `action(E)` or `auto(E)`), their preconditions and effects (`prec(F,E)` and `effect(E,F)`), as well as the possible priorities between them (`prio(E1,E2)`). To account for multiple agents, action events are represented with function term `act(Ag,ActName)` where `Ag` is the agent executing the action and `ActName` is a function term representing the action type and its parameter. The *initial situation* defines the fluents that are initially true, using a number of facts of the form `initially(F)`. It is also used to specify the vocabulary by declaring the objects that are present (and their type) and the positive fluents that can be formed upon them (domain predicate `pfluent(F)`). These two parts of the action context correspond in practice respectively to PDDL domain specification and problem specification (McDermott et al., 1998).

Example 1 (Emergency treatment)**.** In a disaster situation, a team of a medic and a fireman reach a zone with 3 victims. All are injured and will get worse if not treated soon. First victim is in a critical state, second one has serious injuries and is stuck under some debris and third one only has moderate injuries so far. The fireman can extract a victim from the debris, but then the victim will start bleeding which will cause her death if she does not received some blood just after. The medic can heal the injury of an accessible victim (not stuck under debris). If the victim was already in a critical state, she will remain weakened. By using some blood transfusion (action 'support') during her intervention, the medic can avoid death by bleeding out if the victim was bleeding and allow recovery from a weakened state. In our scenario, the medic first heals victim 1 without using her blood pouch while the fireman extracts victim 2. Then the medic heals victim 2 (now bleeding) using the blood pouch. At last, the medic heals victim 3, which has by now become critical and will remain weakened.

We model the actions as follows[1]*: First, the medic can perform actions (i)* `heal(V,I)`*, removing the injury* `I` *of a victim* `V`*, which is possible when the victim* `V` *is alive, not stuck and has injury* `I`*. and (ii)* `supp(V,I)`*, using blood pouch on an injured victim, which is possible when the victim is alive, unstuck, injured and the medic still has a blood pouch. Its effects are to remove the* `weakened` *and* `bleeding` *state of the victim. The fireman can only perform action* `extr(V)`*, extracting a victim who is stuck in some debris. It is possible whenever* `V` *is stuck and has the effects* `neg(stuck(V)` *and* `bleeding(V))`*, as extracting the victim will release the pressure on its wounds.*

We use automatic events here to handle the automatic progression of the state of the victim when not attended: `worsen` *will worsen the state of the injury (from moderate to serious, serious to critical) of a victim at each time step in which the victim is still injured (and this event is not overtaken by some action tending to the victim). When the injury is serious,* `worsen` *will additionally weaken the victim, and it will kill her when the injury is already critical. Death can also occur when a victim is left bleeding, which is modeled by automatic event* `dieB`*. Two additional automatic events are used to emphasize final state of surviving victims :* `save(V)` *registers that a victim survives (when it is alive, and neither stuck, injured or bleeding) and* `stwk(V)` *indicates that a victim will stay weakened when it has been healed while her* `weakened` *status has not been removed.*

A *scenario* can be defined as a set of couple (A, T) indicating that action A is performed at time T. We represent it as a set of ground instances of the `performs(A,T)` predicate. A scenario is *correct* if all performed actions actually occur at the given time in the enfolding of events simulated by the event motor given the action context.

Example 2. *The proposed scenario s_0 is given as :*

```
performs(act(m,heal(v1,crit)),0).  performs(act(f,extr(v2)),0).
performs(act(m,heal(v2,crit)),1).  performs(act(m,supp(v2,crit)),1).
performs(act(m,heal(v3,crit)),2).
```

Checking the event trace, we can verify that each of these actions occur at the given time. This scenario is correct for our model. For illustrative purpose, we will also use another

[1]Full code for this paper can be found on `https://gitlab.lip6.fr/sarmiento/CME2021`

scenario s_1 where the medic decides to use the blood pouch at time 0 on the first patient. The second victim then cannot be saved and the medic thus focus at time 1 on victim 3.

```
performs(act(m,heal(v1,crit)),0).  performs(act(f,extr(v2)),0).
performs(act(m,supp(v1,crit)),0).  performs(act(m,heal(v3,serious)),1).
```

2.2 Event motor

The main purpose of the event motor is to infer from the initial situation and the performed actions the set of events that occur and the evolution of the fluents at each step, that is, to produce the event trace of each scenario. This trace is a set of ground instances of predicate `holds(F,T)`, which indicate that (possibly negated) fluent `F` holds at time `T` and predicate `occurs(E,T)`, expressing that event `E` occurs between time `T` and `T+1`.

Events effect axioms These axioms define the principles that govern fluents: a fluent holds at `T` if it was just initiated by an event occurrence at `T-1` (or it was so initially if `T= 0`); a fluent which is true at `T` holds until the occurrence of an event which terminates it ; it is false in all other cases, which is here made explicit for the fluents that appear as negative preconditions.

```
holds(F,0):-initially(F).
holds(F,T):-effect(E,F),occurs(E,T-1),pfluent(F),time(T).
holds(F,T):-holds(F,T-1),{occurs(E,T-1):effect(E,neg(F)}0,time(T).
holds(neg(F),T):-prec(neg(F),_),not holds(F,T),time(T).
```

Events precondition axioms These axioms characterise the behaviour and the triggering of events, defining in a unique way which predicates of type `occurs(E,T)` are verified at each moment. First, we define predicate `poss(E,T)`, which indicates that `E` is possible at `T`, meaning that all its preconditions hold. As several incompatible events can be possible at the same time, we distinguish the triggering of an event (`trigg(E,T)`), that is, the fact that the event *tries* to execute, from its actual occurrence, that is, the fact that it actually occurs. Priorities between incompatible events are given using predicate `prio(E2,E1)` (meaning that `E2` has priority over `E1`). The principles governing the occurrence of events are as follows: an automatic event (resp. an action) is triggered whenever possible (resp. whenever possible and performed); any triggered event occurs unless it is `overtaken` by the occurrence of another event having priority over it.

```
poss(E,T):-{not holds(F,T):prec(F,E)}0,event(E),time(T).
trigg(U,T):-poss(U,T),auto(U).
trigg(A,T):-poss(A,T),performs(A,T),action(A).
ovtkBy(E1,E2,T):-poss(E1,T),occurs(E2,T),prio(E2,E1).
overtaken(E1,T):-ovtkBy(E1,_,T).
occurs(E,T):-trigg(E,T),not overtaken(E,T).
```

Concurrency and event priorities Contrarily to Berreby, Bourgne, and Ganasia (2018), we consider here that multiple events can occur at the same time, provided that they are compatible, meaning that their concurrent execution would not pose any interpretation issue. Typical cases of incompatibility between events are cases when the execution of one would falsify the preconditions of the other or when they produce opposed effects. This corresponds to non-interferring events in PDDL.

Example 3. *Priorities for our example are defined as follows:*

```
prio(act(m,heal(V,I)),worsen(V,I)):-vict(V),inj(I).
prio(act(m,supp(V,I)),dieB(V)):-vict(V),inj(I).
prio(act(m,heal(V,I)),act(m,heal(V2,I2))):-V!=V2,[...].
prio(act(m,supp(V,I)),act(m,supp(V2,I2))):-V!=V2,[...].
prio(act(m,heal(V,I)),act(m,supp(V2,I2))):-V!=V2,[...].
```

Here, healing a given victim would prevent her state from worsening and giving it blood (action support) would prevent her from bleeding out, so these actions are given priority over these automatic events (ensuring they have a chance to falsify their preconditions before they are triggered). Next priorities ensure that the medic cannot heal or support two different victims at the same time, nor support a victim while healing another one. Note at last that when the target `v` *is the same,* `supp(v,I)` *and* `heal(v,I)` *are compatible.*

2.3 Detecting omission

An agent, broadly speaking, is an entity with the power to act; exercising this capacity makes agents liable to blame or praise, both in ethics and in the law. Yet this capacity is not just a matter of performed actions: surely we are to blame if we choose not to rescue a drowning child. Responsibility therefore also pertains to the power to *not* act. Whether there is a fundamental moral difference between actions and omissions is an important point of debate within moral philosophy (see Bennett and Bennett (1998); Foot (1985)). As such, it is critical to be able to model them both separately in our agnostic architecture.

The meaningful fact of omitting to act only occurs when *acting is possible*. One cannot omit to act if there is no act to omit (Weiner, 1995). As such, we state that an omission occurs when an action is possible, not overtaken, and not performed. Both an action or such an omission constitute a *volition* — i.e. a decision made. However, the fact that an action is overtaken does not necessarily imply that acting was not possible. If it is overtaken by another action, different choices could have been made to perform it. We thus also consider such case as forced omission. It is not a volition per se, but rather the consequence of another volition (the choice to do the other action). Given an action `act(D,X)`, its omission is denoted by `omit(D,X,A)` where `A` is either constant `no` or an action `A` that overturned `act(D,X)`.

```
occurs(omit(D,X,no),T):-A=act(D,X),poss(A,T),
          not overtaken(A,T),not performs(A,T).
occurs(omit(D,X,A2),T):-ovtkBy(act(D,X),A2,T),action(A2).
```

3 Causality analysis

Causality is a central element of ethical decision making as it links an action with the changes and events it provokes, and so infer its consequences, which is the basis of consequentialist approaches. Moreover, it is also instrumental to defining what is a means to an

end, a distinction used both by Kant and the doctrine of double effect. Assessing causal links within a particular case, as we do here, is called *actual causality*. Main approaches are based on counter-factual (Pearl, 2003) or regularity-based (Wright, 2011).

In this framework, to take advantage of the structured information provided by the action model, distinguishing condition and transition as well as actions and omissions, we use event-based causality (Berreby, Bourgne, and Ganasia, 2018). We focus here on supporting relations, that is, stating that the *occurrence of an event* resulted in the occurrence of another. Identifying when the occurrence of an event ensures that another event does not happen, is left for future work. Occurrences of events are reified with function term `o(E,T)` (corresponding to `occurs(E,T)`). Likewise, `h(F,T)` represents the fact that fluent `F` holds at time `T`. A causal relation will thus be represented by ternary predicate `r(R,A,B)` where `R` is a keyword qualifying the kind of causal relation, and `A` and `B` are either of the form `o(E,T)` or `h(F,T)`.

Causing and enabling An obvious starting point to define causes from an action model is to consider that an event *causes* its effects. Factoring in inertia, by considering that `h(F,T)` causes `h(F,T+1)` when both are true, we can directly define that the occurrence of an event at time `T1` will cause a (possibly negated) fluent `F` to hold at a later time `T2` when `F` is one of its effects and it remained true from `T1+1` to `T2`.

```
r(causes,o(E,T1),h(F,T2)):- occurs(E,T1),holds(F,T2),effect(E,F),
            {not holds(F,T):T>T1,T<T2, time(T)}0, T2>T1.
```

We shall then consider that a causal link exists when an occurrence of event causes at least one precondition of another event to hold at the time this second event occurs. As causing something by a deterministic chain of events is different from giving another agent the opportunity to cause another thing by its actions, we distinguish the relation `causes`, where an event causes a precondition of an automatic event and the relation `enables`, where an agent causes a precondition of an action, making it possible for the agent of this action to perform it.

```
r(causes,h(F,T),o(U,T)) :-holds(F,T), prec(F,U), occurs(U,T), auto(U).
r(enables,h(F,T),o(A,T)):-holds(F,T), prec(F,A), occurs(A,T), action(A).
```

At last, we can chain these two steps by stating that an event `E1` will cause (resp. enable) another event `E2` if it causes a fluent `F` which causes (resp. enables) `E2`. Moreover, causing is 'transparently' transitive, meaning that an event `E1` that causes an event `E2` that itself causes or enables a third event `E3` will be considered to cause or enable that third event `E3`.

```
posRel(causes;enables).
r(R,o(E1,T1),o(E2,T2)) :- r(causes,o(E1,T1),h(F,T2)),
     r(R,h(F,T2),o(E2,T2)), T1<T2, posRel(R).
r(R,o(E1,T1),o(E2,T2)) :- r(causes,o(E1,T1),o(E3,T3)),
     r(R,o(E3,T3),o(E2,T2)), T1<=T3, T3<=T2, posRel(R).
```

Causal effects of omissions While they have no operational effect, causally, omission preserves the truth values of the fluents that would have been affected by the omitted action. If an agent chose not to save a children that is drowning, this model of the omission's effect would result in concluding that it contributes to this drowning (i.e. it is considered as one of the causes of this death) by failing to prevent it. In practice, an omission only causes the preservation of the fluents that would have been affected by the action if they are not otherwise changed by another concurrent event. To reflect this, we adapt the first causality rule for omission, using predicate `compl` to get the complementary of a fluent literal. Moreover, when the omission is not a volition per se, but rather a consequence of the volition to do the other action, we explicitly state that this other action causes the omission.

```
compl(F,neg(F)):-pfluent(F).     compl(neg(F),F):-pfluent(F).
r(causes,o(omit(D,X,A),T1),h(F,T2)):- occurs(omit(D,X,A),T1),
            holds(F,T2),compl(F,neg(F)),effect(act(D,X),neg(F)),
            {not holds(F,T):T>T1,T<T2,time(T)}0, T2>T1.
r(causes,o(A,T),o(omit(D,X,A),T)):-occurs(omit(D,X,A),T), A!=no.
```

At last, when omitting to perform an action, one allows the events that would have been overtaken by this action to happen. This can be seen as a case of failing to prevent something. Indeed, if considering opposing causal relation such as prevention, we can say that performing an action that overtakes an automatic event will prevent the occurrence of this event at this time step (at least). By omitting such an action, one fails to prevent the event, which thus actually happens. Such a relation is a supporting relation as it links events that happen in the unfolding. Ethically, however, allowing something to happen and causing it do not carry the same weight, so we model this case as a new causal relation: `allows`. This relation will have the same transitivity property as `enables` (which amount to declaring it as a `posRel`).

```
r(allows,o(omit(D,X,A),T),o(U,T)) :- occurs(omit(D,X,A),T),
    occurs(U,T), prio(act(D,X),U), auto(U).
posRel(allows).
```

Modelling causal effects of plans Since ethical assessment rely for a great part on the consequences of the choice being assessed, in order to assess scenario with multiple decisions, we can consider that selecting a scenario corresponds to the choice of performing each of its actions. In a collaborative setting where agents can communicate before hand, this would be a collective choice. We can model this by denoting by constant `plan` the choice to commit to the scenario being unfolded and consider that this choice causes all of the volitions in the scenario, meaning that a plan causes each of the actions that are performed, but also each of the omissions that do not result from other choices. The following rules reflect this idea.

```
r(causes,plan,o(A,T)):-occurs(A,T), performs(A,T).
r(causes,plan,o(omit(D,X,no),T)):-occurs(omit(D,X,no),T).
```

Example 4. *Figure 1 illustrates the unfolding of our first scenario and the causal relations between the events occurring in the event trace. To be more concise, occurrences of actions (*`o(act(D,X),T)` *are denoted as* a^T_X*. Likewise, omission* `(o(omit(D,X,_),T)` *and automatic event* `o(E,T)` *are denoted respectively* o^T_X *and* u^T_E*, while constant* `plan` *is just denoted as P.*

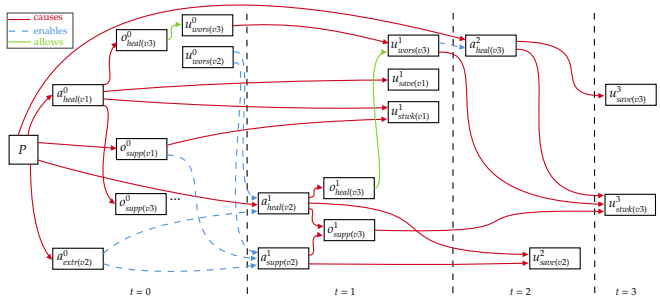

Figure 1: Causal relations between event occurrences of the running example.

We can see that in this scenario, all victims are saved, but both victim 1 and victim 2 stay weakened. The causal trace correctly analyses that (i) healing victim 1 at time 0 will cause this victim to be saved ; (ii) the choice not to support victim 1 at time 0 causes this victim to stay weakened, but this omission also enables the action `supp(v2,crit)` *at time 1, which causes* `o(save(v2),2)`*; (iii) omitting to heal the victim 3 at time 0 and 1 each time allows the states of victim 3 to worsen, which in turn causes this victim to stay weak in the end. As we do not model opposing relation here, we cannot however causally link the action* `supp(v2,crit)` *at time 1 to the fact that we cannot support victim 3 at time 2 (which is not an omission as its preconditions are not met). This would be a case of exclusion (one action making a future one impossible by falsifying its preconditions).*

4 Ethical assessment

The different models given in the previous sections allow an agent to foresee the unfolding of events and derive the consequences of its actions, but these models do no contain any ethical considerations. To assess the ethical admissibility of actions, there is a need to characterize the decision or their outcome. Berreby, Bourgne, and Ganascia (2017) propose two elements for the ethical layer. First, a *model of the Good* performs an evaluation of the Good produced by the actions. *Theories of the Right* can then rely on this evaluation to assess what is or not permissible in a given context.

Model of the Good A model of the Good must evaluate the atomic good and bad produced by each event in order to qualify and eventually quantify them. It is not directly affected by concurrency or omissions. In our running example, we consider two modalities : the right to live, which is ensured by `save(V)` and violated by any event causing death, and the right to be in good health, which is violated by `stwk(V)`. We quantify these by giving respectively weight of 10 and 2 for the right to live or to be in good health.

```
wGood_elem(save(V),V,life,10):-event(save(V)).
wBad_elem(E,V,life,10):-effect(E,neg(alive(V))).
wBad_elem(stwk(V),V,health,2):-event(stwk(V)).
wGood(o(S,E,T),V,M,N):- wGood_elem(E,V,M,N); occurs(S,E,T).
wBad(o(S,E,T),V,M,N):- wBad_elem(E,V,M,N), occurs(S,E,T).
```

Theories of the Right The model of the Right is tasked with determining which actions or set of actions are right according to the circumstances of their execution. Each theory of the Right then represents a given ethical principle (the different theories being given by `th(P)`). This model can infer impermissible actions according to each of its ethical principles using predicate `imp(P,A)`, which indicates that volition occurrence `A` is inadmissible (or impermissible) according to theory `P`. Ethical principles usually focus, in classical dilemma, on defining which *action* you should do. But as actions can be part of a greater collective plan, we will here assess directly the whole scenario, using constant `plan`, which represents the volition to perform the whole set of actions decided upon, causing each action and deliberate omission of the scenario.

While most deontological theories define *satisficing* principles, i.e. principles that can assess the permissibility of an action or a scenario in itself, without considering alternatives courses of actions, some consequentialistic approaches prescribe *maximising* principles. Rather than assessing whether an option is good enough, they focus on finding the best options given the possible alternatives. To model such principles, we must compare different scenarios within the same program. To each deterministic unfolding of a scenario corresponds a unique simulation `S` that we add to each occurrence of events : `o(E,T)`, `plan` and `occurs(E,T)` appearing in simulation `S` are rewritten as respectively `o(S,E,T)`, `plan(S)` and `occurs(S,E,T)` through some post processing gathering the traces of all scenarios. A permissible scenario is then one that is not impermissible: `per(P,plan(S)):-sim(S), not imp(P,plan(S)), th(P).`

Some theories of the Right We propose three theories to illustrate this architecture. The first two of these theories are adapted from Berreby (2018) and the last one is adapted from Lindner and Bentzen (2018). This first sample could be expanded easily by adapting more principles from such sources.

Benefit Cost Analysis and Act-utilitarism. First two theories are consequentialistic, with the first one being an example of a satisficing consequentialistic approach while the second one is maximising. Our first step is thus to gather and aggregate individual good and bad of all consequences of one's actions, considering that each causal relations R can be given a weight f_R : $w(A) = \Sigma_{R \in \texttt{posRel}}(\Sigma_{(k,x,m,e) \in G_R(A)} k.f_R - \Sigma_{(k,x,m,e) \in B_R(A)}(f_R \times k)$

where $G_R(A) = \{(k,x,m,e)|_{\texttt{wGood}(k,e,m,k),\texttt{r}(R,A,e)}\}$, B_R is similar to G_R using instead `wBad`, $f_{\texttt{causes}} = f_{\texttt{allows}} = 10$ and $f_{\texttt{enables}} = 0$. This is translated to ASP rules with sum aggregates deriving `weightAct(`A`,`$w(A)$`)`. Using this aggregation, we define the benefit cost analysis principle `benCost` (one should act in a way that it brings about more good than bad) and the act-utilitarism principle `actUti` (one should act in a way that maximises the overall good).

```
th(benC). th(actU).
imp(benC,plan(S)):-weightAct(plan(S),N),number(N),N<0,sim(S).
imp(actU,plan(S)):-weightAct(plan(S),N1), weightAct(plan(S2),N2),
      N1<N2,sim(S;S2).
```

Kant 2nd categorical imperative. At last, we propose an adaption of Kant's second categorical imperative based on the formalisation by Lindner and Bentzen (2018) (using their first reading of "means"). This principle states that you should "act in such a way that you treat humanity [...] never merely as a means to and end, but always at the same time as an end." This requires additional ethical specification of the moral patient (`mPatient(P)`), which event affects them and how (`affect(EvOcc,P,K)`, where `EvOcc` is an event occurrence, `P` a moral patient and `K`$\in \{-1,1\}$ indicates whether it is affected negatively or positively) and what is the aim of the scenario (`aim(plan(S),EvOcc)`). In our example, moral patients are the victims, affected by each event having them as parameters and the aim of each scenario is to save them. Using these, the theory itself defines what is an end, a means, before prohibiting scenarios that consider some moral patient as a mean and not an end.

```
th(kant(1;2)).
end(A,X):-affect(G,X,1), aim(A,G), {affect(G2,X,-1):aim(A,G2)}0.
means(A,X):-r(R1,A,E),affect(E,X,_),r(R2,E,G),aim(A,G),posRel(R1;R2).
imp(kant(N),plan(S)):-means(plan(S),X),not end(plan(S),X),mPatient(X).
```

Example 5. *Going back to our example, considering our basic scenario s_0 and only the alternative scenario s_1 where the medic support victim 1 in the first time step (resulting in two full remissions but one death), the program assess that both scenarios are permissible for satisficing principles (*`benCost` *and* `kant`*), while only the first is permissible according to* `actU`*. Should we consider the* `weakened` *state to represent something really serious (for example coma), giving it a much higher penalty (e.g. 12), the act-utilitarian approach would then prefer the second scenario, making the first one impermissible.*

5 Conclusion

We presented here an architecture to structure ethical reasoning, and especially the task of assessing moral permissibility of some scenarios with respect to different ethical principles. The ACE framework allows us to separate modelling concerns and provide a clear picture of the reasoning involved in an ethical decision. The proposed implementation, presented as a machinery for assessment scenario defined beforehand can easily be geared toward planning by using choice axiom to generate scenarios (and integrity constraints to ensure they are correct). Even without using the underlying implementation in ASP, we believe this separation in three layers to be a useful methodological tool to model ethical principles. More specifically, we discussed here how to deal with explicit representation of omission in a concurrent setting and proposed a simple way to assess plans by evaluating the collective decision to commit to a plan as a meta action causing all the volitions involved in the scenario. As most works in normative ethics focus on the assessment of a single decision within a context, this allows us to lift up the modelling of multiple actions to a single decision that can be modeled accordingly. We left for future work a more detailed analysis of the interactions between omission and opposing causal relations.

References

Bennett, J., and Bennett, J. F. 1998. *The act itself*. Oxford University Press.

Berreby, F.; Bourgne, G.; and Ganascia, J.-G. 2017. A declarative modular framework for representing and applying ethical principles. In *AAMAS*, 96–104. IFAAMAS.

Berreby, F.; Bourgne, G.; and Ganasia, J.-G. 2018. Event-based and scenario-based causality for computational ethics. In *AAMAS*, 147–155. IFAAMAS.

Berreby, F. 2018. *Models of ethical reasoning*. Ph.D. Dissertation, Sorbonne Université, EDITE, LIP6, Paris.

Foot, P. 1985. Morality, action, and outcome. *Morality and objectivity* 23–38.

Lindner, F., and Bentzen, M. 2018. A formalization of kant's second formulation of the categorical imperative. DEON 2018, Utrecht, Netherlands.

Lindner, F.; Bentzen, M. M.; and Nebel, B. 2017. The HERA approach to morally competent robots. In *IROS*, 6991–6997. IEEE.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL-the planning domain definition language.

Mueller, E. T. 2008. Event calculus. In *Handbook of Knowledge Representation, Foundations of Artificial Intelligence*, volume 3. Elsevier. 671–708.

Pearl, J. 2003. Causality: models, reasoning, and inference. *Econometric Theory* 19:675–685.

Tolmeijer, S.; Kneer, M.; Sarasua, C.; Christen, M.; and Bernstein, A. 2021. Implementations in Machine Ethics: A Survey. *ACM Computing Surveys* 53(6):132:1–132:38.

Weiner, B. 1995. *Judgments of responsibility: A foundation for a theory of social conduct*. guilford Press.

Wright, R. 2011. The NESS Account of Natural Causation: A Response to Criticisms. In *Perspectives on Causation*. Hart Publishing. 38.