Cadre déclaratif modulaire d'évaluation d'actions selon différents principes éthiques

Fiona Berreby , Gauvain Bourgne , Jean-Gabriel Ganascia

CNRS & Sorbonne Universités, LIP6, 4 place Jussieu 75005 Paris, France
{fiona.berreby,gauvain.bourgne,jean-gabriel.ganascia}@lip6.fr

RÉSUMÉ. *Cet article examine l'utilisation de langages de haut niveau dans la conception d'agents autonomes éthiques. Il propose un cadre logique nouveau et modulaire pour représenter et raisonner sur une variété de théories éthiques, sur la base d'une version modifiée du calcul des évènements, implémentée en Answer Set Programming. Le processus de prise de décision éthique est conçu comme une procédure en plusieurs étapes, capturée par quatre types de modèles interdépendants qui permettent à l'agent d'évaluer son environnement, de raisonner sur sa responsabilité et de faire des choix éthiquement informés. Notre ambition est double. Tout d'abord, elle est de permettre la représentation systématique d'un nombre illimité de processus de raisonnements éthiques, à travers un cadre adaptable et extensible. Deuxièmement, elle est d'éviter l'écueil trop courant d'intégrer directement l'information morale dans l'engin de raisonnement général sans l'expliciter, alimentant ainsi les agents avec des réponses atomiques qui ne représentent pas la dynamique sous-jacente. Nous visons à déplacer de manière globale le processus de raisonnement moral du programmeur vers le programme lui-même.*

ABSTRACT. *This paper investigates the use of high-level action languages for designing ethical autonomous agents. It proposes a novel and modular logic-based framework for representing and reasoning over a variety of ethical theories, based on a modified version of the Event Calculus and implemented in Answer Set Programming. The ethical decision-making process is conceived of as a multi-step procedure captured by four types of interdependent models which allow the agent to assess its environment, reason over its accountability and make ethically informed choices. The overarching ambition of the presented research is twofold. First, to allow the systematic representation of an unbounded number of ethical reasoning processes, through a framework that is adaptable and extensible. Second, to avoid the common pitfall of too readily embedding moral information within computational engines, thereby feeding agents with atomic answers that fail to truly represent underlying dynamics. We aim instead to comprehensively displace the burden of moral reasoning from the programmer to the program itself.*

MOTS-CLÉS : *éthique computationelle, answer set programming, calcul d'évènements, raisonnement sur l'action et le changement.*

KEYWORDS: *computational ethics, answer set programming, event calculus, reasoning about actions and change.*

DOI:10.3166/RIA.32.479-518

1. Introduction

L'étude de la morale d'un point de vue computationnel a attiré l'intérêt croissant de chercheurs en intelligence artificielle (Anderson, Anderson, 2011). L'autonomie croissante des agents artificiels et l'augmentation du nombre de tâches qui leur sont déléguées nous incitent à aborder leur capacité à traiter les restrictions et les préférences éthiques, que ce soit dans leur propre structure interne ou pour des interactions avec des utilisateurs humains. Des domaines aussi variés que la santé ou le transport posent des problèmes éthiques qui sont en ce sens particulièrement pressants, car ils peuvent exiger de la part des agents des prises de décisions dont les conséquences sont immédiates ou lourdes. L'éthique computationnelle peut aussi nous aider à mieux comprendre la morale et raisonner plus clairement sur les concepts éthiques qui sont employés dans des domaines philosophiques, juridiques et technologiques. Dans ce contexte, notre objectif est de fournir une architecture modulaire qui permette la représentation systématique et adaptable de principes éthiques. Pour ce faire, nous présentons un ensemble de modèles qui, ensemble, permettent à l'agent d'évaluer son environnement, d'intégrer des règles éthiques et de déterminer à partir de la mise en œuvre de ces règles soit un plan d'action, soit une évaluation du comportement d'autres agents. Ceux-ci sont implémentés en Answer Set Programming, sur la base d'une version modifiée du calcul des évènements.

L'article est structuré comme suit. Nous commençons par présenter les concepts philosophiques pertinents ainsi que les travaux connexes (section 2), puis nous présentons l'architecture du cadre (section 3). Ensuite, nous définissons et discutons de chaque modèle (sections 4 à 7) en les illustrant par deux exemples, puis nous discutons des résultats obtenus pour différents principes sur ces deux exemples (section 8), et concluons (section 9). Le code des exemples fournis dans cet article peut être téléchargé (`https://gitlab.lip6.fr/bourgneg/RIA-CadreEthique/`).

2. Contexte

2.1. Théories éthiques

L'étude de l'éthique est l'étude des croyances que les gens peuvent ou devraient avoir pour contrôler leur comportement. Une classification tripartite standard divise le champ entre la *méta-éthique*, qui concerne le statut ontologique des concepts éthiques, l'*éthique appliquée*, qui concerne l'application des règles morales à des environnements particuliers, et l'*éthique normative*, qui traite de la définition, de la comparaison et de l'explication de conceptions éthiques (Bringsjord, Taylor, 2012). Le présent travail revêt de l'importance pour l'éthique appliquée, en ce sens qu'il propose un schéma de conception d'agents artificiels contraints par l'éthique agissant dans des domaines appliqués, mais aussi pour l'éthique normative car son but est de modéliser les processus qui sous-tendent la prise de décision éthique normative, avec la possibilité de confronter différentes perspectives. Nous nous concentrons sur deux de ses principales branches: l'éthique conséquentialiste et l'éthique déontologique.

Le Bien et Le Juste

Les éthiques conséquentialistes s'articulent autour de l'idée que les actions doivent être évaluées en fonction de leur conséquences, et ne peuvent être justes ou injustes qu'en vertu de ce qu'elles produisent. Une action moralement juste est celle qui produit un bon, ou le meilleur, état de choses. Or, afin de déterminer la justesse d'une action, les conséquentialistes doivent d'abord établir ce qui constitue un *bon* état de choses, c'est-à-dire déterminer ce qu'on appelle plus largement 'Le Bien' (Alexander, Moore, 2016). Cela leur permet ensuite d'affirmer que des actions font partie du 'Juste' dans la mesure où elles augmentent le Bien. Les théories conséquentialistes suivent donc, s'appuient et finalement dépassent, les théories du Bien. Les désaccords entre conséquentialistes sur ce qui constitue le Bien ont engendré diverses traditions et doctrines conséquentialistes. L'*utilitarisme* voit le Bien résulter de la maximisation du bien-être collectif, l'*altruisme éthique* du bien-être des autres, l'*égoïsme éthique* de l'intérêt personnel, l'*utilitarisme des droits* du respect des droits individuels.

Les théories déontologiques (du grec *deon*, "devoir") prétendent que la valeur morale d'une action est déterminée -au moins en partie- par une caractéristique intrinsèque de l'action. Habituellement, cette caractéristique est une obligation ou une interdiction. Par exemple, une règle déontologique peut indiquer que le mensonge est contraire à l'éthique, ce qui implique que tout énoncé qui contient un mensonge est interdit. Parce qu'une action est jugée juste ou non en fonction de sa conformité avec une norme ou un devoir, son évaluation éthique est au moins partiellement indépendante de ses conséquences. Le Juste est ici prioritaire sur le Bien: une action peut être injuste pour le déontologue même si elle maximise le Bien, et juste même si elle le minimise. Les tentatives de définition du Bien seront désormais appelées *théories du Bien*, et les tentatives de définition du Juste, qu'elles soient conséquentialistes ou déontologiques, seront appelées *théories du Juste*.

2.2. Éthique computationnelle

L'éthique computationnelle est un champs d'étude vaste et en expansion. Depuis les premiers travaux au milieu des années 2000, il a suscité durant la dernière décennie un intérêt croissant dans la recherche en intelligence articifielle. Certains travaux s'intéressent de manière large aux principes nécessaires à l'élaboration de systèmes éthiques. Dignum (Dignum, 2017), par exemple, identifie comme principes fondamentaux pour l'élaboration de tels systèmes la transparence et deux types de responsabilité (*accountability* et *responsibility*). Cette approche d'*éthique par conception* a donné lieu à de nombreux travaux (Dodig Crnkovic, Çürüklü, 2012 ; Bryson, 2018). Il existe aussi des approches de vérification formelle (Dennis *et al*., 2016) pour s'assurer des propriétés de tels systèmes. Nos travaux s'inscrivent cependant plutôt dans le contexte de la prise de décision éthique, où il s'agit de déterminer des choix respectant certains principes éthiques. L'étude de la décision éthique est généralement abordée de deux points de vue, selon que l'on adopte une approche ascendante ou descendante (Wallach *et al*., 2008).

Les *approches ascendantes* essaient de permettre des décisions éthiques en s'appuyant sur un certain nombre d'exemples ou de cas. Ce sont des approches descriptives, tentant d'apprendre ou d'inférer les bonnes décisions en fonction d'un ensemble de cas fournis en exemples. L'objet est ainsi de reproduire un comportement observé, qu'il s'agisse des annotations d'un expert ou d'intuitions générales collectées dans la population, comme dans le cas des données issues du projet MIT Moral machines (`http://moralmachine.mit.edu`) utilisées notamment par (Noothigattu *et al*., 2018).Les travaux dans ce domaine s'appuient généralement sur des techniques de raisonnement par cas (McLaren, 2006), de raisonnement analogique (Blass, Forbus, 2015) ou d'apprentissage, avec généralement des approches statistiques comme l'apprentissage par renforcement (Wu, Lin, 2018). Si la plupart de ces systèmes ne construisent pas de représentations explicites ou interprétables des principes éthiques appris, il existe des exceptions telles que *GenEth* (Anderson, Anderson, 2014), qui s'appuie sur des connaissances extérieures pour guider un apprentissage par programmation logique inductive donnant lieu à des règles explicites. A l'inverse, nous préférons une *approche descendante*, appuyant la prise de décision éthique sur la modélisation de principes éthiques exprimés en philosophie. Il s'agit là d'une approche *prescriptive* : l'objet n'est pas de simuler le comportement d'un humain, mais de reproduire les recommendations de philosophes. Certains travaux s'intéressent à l'intégration de tels principes dans l'architecture d'un agent, que ce soit par l'intégration de normes morales (Tufiş, Ganascia, 2015 ; Serramia *et al*., 2018) ou de règles morales et de procédure d'évaluations éthiques (Cointe *et al*., 2016), mais nous nous focalisons ici sur les approches logiques et la représentation de ces théories éthiques. Dans ce cadre, les principaux travaux s'appuient sur des logiques déontiques (Arkin, 2009 ; Govindarajulu, Bringsjord, 2017), des variantes de la logique STIT (Lorini, 2012 ; Lorini *et al*., 2014) ou de la programmation logique, se basant sur des logiques non-monotone comme l'Answer Set Programming (Ganascia, 2007 ; 2015) ou la programmation logique prospective (Pereira, Saptawijaya, 2007 ; 2017).

La logique STIT, définissant la capacité d'un groupe d'agent à assurer un certain état de choses (*see to it that*), offre une approche générale et pertinente pour évaluer conséquences et responsabilité qui s'applique naturellement à des contextes multi-agents (Lorini *et al*., 2014). Cette approche tend toutefois à se focaliser sur les états atteints et atteignables plutôt que sur les actions qui les sous-tendent, ce qui est très adapté aux approches conséquentialistes, mais rend plus délicate la modélisation des approches déontologiques et certaines différentiations pertinentes comme celle entre action et inaction, ou effets directs et indirects. Certaines variantes, tels que la logique dynamique des attitudes mentales et actions combinées (DL-MA)(Lorini, 2012), rétablissent néanmoins une identification explicite des actions intervenant dans les transitions. Celle-ci intègre aussi des réprésentations explicites de la désirabilité et de l'idéalité, en les associant à un monde donné, mais sans discuter des éventuels liens entre ces valeurs et les états ou l'historique du monde associé. Pourtant, le processus de décision proposé se base entièrement sur ces valeurs : c'est donc dans leur détermination en fonction des autres éléments que se situerait le coeur d'une formalisation de principes éthiques. De la même façon, de nombreux modèles intégrent directement

l'information éthique au sein du processus de prise de décision de l'agent, sans pour autant générer un raisonnement moral. En tachant de décomposer et de rendre explicite les mécanismes de détermination du Bien et du Juste, qu'ils soient généraux ou spécifiques à une situation, l'approche que nous proposons ici vise à diminuer ce biais.

Un autre exemple pertinent d'approche logique, basée ici sur des aspects déontiques, est le calcul des évènements cognitif et déontique ($\mathcal{DCEC}$) (Govindarajulu, Bringsjord, 2017), que les auteurs utilisent pour modéliser formellement la doctrine du double effet (DDE). Ils se basent toutefois sur des notions de prouvabilité qui ne sont pas représentables dans leur langage : les agents ne peuvent donc explicitement manipuler les conclusions d'admissibilité ou non des différentes actions selon la DDE. Il est toutefois notable de signaler que cette formalisation présente des avantage de généralité similaire à nos travaux précédents (Berreby *et al*., 2015), s'appuyant sur le formalisme du calcul des évènements pour présenter une formalisation indépendante de la situation. Ce n'est pas toujours le cas, l'analyse se focalisant parfois sur l'expressivité du langage et l'adéquation des mécanismes de preuve proposés sans fournir de formalisation explicite et générale des principes modélisés. Ces derniers sont alors simplement illustrés sur des dilemmes classiques, réduit à une expression simplifiée qui peut cacher des raisonnement non triviaux. Par exemple, en utilisant une logique prospective, Pereira et al. (Pereira, Saptawijaya, 2007) modélisent une règle déontologique qui prohibe le meurtre intentionnel par la règle *falsum* $\leftarrow$ *intentionalKilling*. Or ils déterminent si *intentionalKilling* vaut pour une action en indiquant atomiquement si cette action l'implique, utilisant des règles de la forme *intentionalKilling* $\leftarrow$ *end(A,iKill(Y)).* où A est l'action évaluée. Le problème avec cette approche est que l'évaluation éthique est *indiquée* par des énoncés spécifiques à l'action, plutôt qu'*extraite* par une forme de compréhension de l'environnement et des règles éthiques en place. Il n'y a pas de représentation de la causalité, de sorte que l'action et ses conséquences ne sont pas liées dynamiquement : leur relation est déclarée plutôt que déduite. Aucune notion de responsabilité éthique ne peut être élaborée sur cette base. En outre, les règles données manquant de puissance expressive, un nouveau programme est nécessaire pour modéliser chaque nouveau cas. Ces formalismes ne peuvent donc pas contraster différentes théories, ni expliciter leurs hypothèses.

2.3. Programmation par Answer Set

La programmation par Answer Set est une forme de programmation logique déclarative avec négation par l'échec adaptée à la représentation de différents problèmes d'Intelligence Artificielle et aux raisonnements non monotones. Développée à partir des fondations posées par Gelfond et Lifschitz (1988 ; 1991), elle a ses racines dans Prolog et tire partie des progrès des solveurs de satisfiabilité propositionnelle modernes, unifiant différents formalismes non-monotones la précédant. Ses implémentations reposent en général sur un un processus en deux temps, avec une première étape d'instantiation de programmes logiques encodés en programmes disjunctifs étendus (EDP) qui précède une étape de résolution utilisant des techniques de recherche de modèles propositionnels pour extraire les modèles stables, ou Answer Sets. En effet, la

sémantique des modèles stables est définie sur des programmes complètement instantiés : un programme avec variables est considéré comme une écriture compacte de son instantiation, ayant les mêmes Answer Sets. Typiquement ces Answers Sets correspondent aux différentes solutions du problème modélisé par le programme (Lifschitz, 2008). Ils forment des ensembles de conclusions cohérentes décrivant les différentes alternatives rationnelles qui peuvent être inférées des règles et faits fournis. Intuitivement, chaque modèle stable est un modèle minimal du programme dont chaque atome est justifié par au moins une règle qui le produit.

Nous donnons ici quelques rappels syntaxiques et sémantiques en se basant sur la syntaxe de `clingo 4.4.0`. Un programme logique normal est constitué d'un ensemble de règles de la forme :

$$A \texttt{:-} B_1, \ldots, B_k, not\ B_{k+1}, \ldots, not\ B_n \texttt{.}$$

où $A, B_1, \ldots, B_n$ sont des atomes de la forme $p(t_1, \ldots, t_q)$ où p est un prédicat d'arité q et $t_1, \ldots, t_q$ sont des termes instantiés.Si on désigne par r la règle ci-dessus, ses différents élément sont sa tête $head(r) = \{A\}$ et so, corps $body(r) = \{B_1, \ldots, B_k, not\ B_{k+1}, \ldots, not\ B_n\}$, comprenant son corps positif $body^+(r) = \{B_1, \ldots, B_k\}$ et son corps négatif $body^-(r) = \{B_1, \ldots, B_k\}$. Une règle sans corps est un *fait* et se note « A . ». Une règle sans négation par l'échec est dite *positive*.

Un ensemble d'atomes S *satisfait* une règle r ssi la satisfaction de son corps implique la satisfaction de sa tête, c'est-à-dire ssi S satisfait $head(r)$ ou S ne satisfait pas $body(r)$, où S satisfait $head(r)$ ssi $head(r) \cap S \neq \emptyset$ (soit $A \in S$ pour une règle normale) et S satisfait $body(r)$ ssi il satisfait chaque atome de son corps positif mais aucun atome de son corps négatif, soit $body^+(r) \subseteq S$ et $body^-(r) \cap S = \emptyset$ (un atome classique B étant satisfait par S ssi $B \in S$). Un *modèle* d'un programme P est un ensemble d'atomes satisfaisant toutes ses règles. Dans le cas d'un programme normal positif P, il existe un unique modèle minimal pour l'inclusion d'ensemble, c'est-à dire un modèle inclus dans tous les modèles de P, que l'on peut obtenir par application successives des règles depuis les faits jusqu'à saturation. Cet unique modèle minimal est appelé le *modèle stable* de P. Dans le cas d'un programme normal quelconque, on se ramène au cas positif en utilisant une transformation appelée la *réduction* : la *réduction* d'un programme P selon un ensemble d'atomes S, notée P^S s'obtient en supprimant des règles de P toutes les occurences de $not\ B$ telles que $B \notin S$, puis en éliminant toutes les règles contenant un $not\ B'$ tel que $B' \in S$. Intuitivement cela revient à remplacer chaque $not\ B$ par vrai ou faux selon que B est effectivement absent de S ou non. Le programme ainsi défini ne contient plus de not, c'est donc un programme positif. Un *Answer Set* est alors défini comme suit : pour tout programme normal P, un ensemble d'atome S est un Answer Set de P ssi S est l'unique modèle stable de la réduction de P selon S. Un programme peut avoir 0, 1 ou plusieurs Answer Sets. On définit l'*inférence sceptique*, notée $\models$, ainsi: étant donné un programme P et un atome A, $P \models A$ ssi A appartient à tous les Answer Sets de P. De plus, on note $Th_{\mathcal{L}}(P)$ la projection des conséquences sceptiques d'un programme P sur un ensemble d'atomes instantiés $\mathcal{L}$: $Th_{\mathcal{L}}(P) = \{p \in \mathcal{L}, P \models p\}$.

Nous utilisons de plus deux autres types de constructs pouvant prendre la place d'un atome dans le corps d'une règle. Les *contraintes de cardinalités* sont de la forme « $\{\tilde{p}(\vec{X}) : cond(\vec{X})\}n$ » où $\vec{X}$ est un ensemble de variables qui n'apparaît que dans cette expression, $\tilde{p}(\vec{X})$ est un atome classique et $cond(\vec{X})$ est de la même forme qu'un corps de règle. Cette expression signifie que le nombre de $\tilde{p}(\vec{X})$ tel que $cond(\vec{X})$ est satisfait doit être inférieur ou égal à n. Une telle expression est satisfaite par un ensemble d'atomes S ssi l'ensemble des $\tilde{p}(\vec{X})$ inclus dans S tels que S satisfait $cond(\vec{X})$ est au plus de cardinal n, soit $|\{\tilde{p}(\vec{X}) \in S, cond(\vec{X}) \text{ satisfait par } S\}| \leq n$. Par exemple `{p(X):q(X,4)}0` est satisfait s'il n'a aucun $p(X)$ tel que $q(X, 4)$ est vrai. De plus, pour calculer des sommes, nous utilisons aussi des *aggrégats* dans des expressions de la forme « $N = \#\mathtt{sum}\{X, \vec{V} : cond(X, \vec{V}, \vec{W})\}$ », où N, X sont des variables entières et $\vec{V}, \vec{W}$ des ensembles de variables quelconques. Une telle expression signifie que N doit être égal à la somme des X sur toutes les valeurs différentes de $(X, \vec{V})$ pour lesquelles il existe des valeurs de $\vec{W}$ telles que $cond(X, \vec{V}, \vec{W})$ est satisfait. Par exemple, pour S contenant $q(1, a, 1)$, $q(1, a, 2)$, $q(1, b, 3)$, $q(2, c, 2)$, « $N = \#\mathtt{sum}\{X : q(X, Y, Z)\}$ » est satisfaite ssi $N = 3$ (1+2, somme des deux valeurs différentes que prend X dans q), et « $N = \#\mathtt{sum}\{X, Y : q(X, Y, Z)\}$ » est satisfaite ssi $N = 4$ (1+1+2, somme des valeurs que prend X pour toutes les valeurs différentes de (X, Y) satisfiant q, soit $(1, a)$, $(1, b)$ et $(2, c)$).

Pour éviter de générer de multiples règles inutiles, la phase d'instantiation détermine les valeurs possibles de chaque variable apparaissant dans une règle à partir des conclusions directes du programmes (basiquement, les faits présents dans le programme et leur propagation unitaire) pour ne conserver que les valeurs des variables susceptibles de satisfaire le corps de la règle. Afin de guider ce processus, il est utile de déclarer à l'aide de faits simples un certains nombres de *domaines* pour typer les variables. Formellement, un domaine prend donc la forme d'un ensemble de termes instantiés décrivant l'ensemble des valeurs que peut prendre une variable. Par la suite, étant donné un domaine de valeurs $\mathcal{X}$ et un prédicat `pred`, on notera par `pred(`$\mathcal{X}$`)` le language (ensemble d'atomes instantiés) définit par l'ensemble $\{$`pred(X), X` $\in \mathcal{X}\}$ (en suivant le même principe pour des prédicat d'arités supérieures sur le produit cartésien de tous les domaines). Ainsi, avec $\mathcal{X} = \{2, 3\}$ et $\mathcal{Y} = \{$`red, neg(red)`$\}$, la notation `q(`$\mathcal{X}$`,`$\mathcal{Y}$`)` représente l'ensemble $\{$`q(2,red), q(3,red), q(2,neg(red)), q(3,neg(red))` $\}$. Nous utilisons les notations `p(`k`..`$n``)`. pour représenter les faits `p(`$k`)`.`p(`$k+1`)`$\ldots$`p(`$n`)`. et `p(a;b;c).` pour représenter `p(a). p(b). p(c).`. Notons que `q((a,b);(c,d))` désigne `q(a,b). q(c,d).` et `t(a,(b;c),d)` correspond à `t(a,b,d). t(a,c,d).`.

L'ASP offre une sémantique bien définie et bénéficie de solveurs efficaces, maintenus et enrichis par une communauté active. Sa gestion de la négation par l'échec permet une représentation aisée de principes naturels en représentation des connaissances (comme les règles avec exceptions) et sa gestion des agrégats et contraintes de cardinalité facilite l'expression déclarative de concepts plus complexes. L'existence de traductions établies du calcul des évènements discrets en ASP (Kim *et al.*, 2009) nous a de plus donné un point de départ concret. Il est à noter que nous ne tirons

pas pleinement partie ici des possibilités de l'ASP en ce sens que le code présenté dans cet article s'appuie sur le déterminisme des phénomènes que nous modélisons pour ne générer au final qu'un seul Answer Set contenant toutes nos conclusions. De ce fait, le code produit dans cet article, moyennant quelques ajustements syntaxiques pourrait théoriquement se traduire, par exemple, en Prolog. Cependant, la possibilité de générer différents Answer Sets face à plusieurs alternatives cohérentes est cruciale pour étendre dans le futur ces travaux à des contextes tels la présence d'autres agents, d'actions aux effets non déterministes ou d'observations incomplètes.

3. Schéma structurel

3.1. Modèles et modularité

La représentation explicite du raisonnement éthique permet à un agent d'informer son processus de prise de décision ou de juger du comportement des autres. Pour y parvenir, il « teste » les actions possibles dans des simulations spécifiques afin d'évaluer leurs conséquences ou leur mérite éthique inhérent. Le résultat de la simulation donne alors un ensemble d'actions acceptables ou inacceptables, qui dicte le comportement à venir. Le cadre présenté ici est concerné par ce processus d'évaluation, plutôt que par l'application de cette évaluation. Il est donc destiné à être par la suite intégré à l'architecture des agents pour intervenir dans les processus de décisions ou de jugement de l'agent. Ce processus d'évaluation éthique est appréhendé comme une procédure en quatre étapes définie par quatre types de modèles interdépendants appliqués successivement : un *modèle d'action*, un *modèle de causalité*, un *modèle du Bien*, et un *modèle du Juste*. Les deux premiers modèles produisent une compréhension entièrement non-éthique du monde, les deux suivants y superposent une compréhension éthique du monde. Le modèle d'action présenté ici, et qui constitue la base du cadre, est basé sur une version modifiée du calcul des évènements à la manière de (Berreby *et al*., 2015). Nous considérons ici un modèle discret et déterministe. La situation est représentée en termes de *fluents*, des propriétés du monde variant dans le temps, et d'*évènements* qui modifient ces fluents.

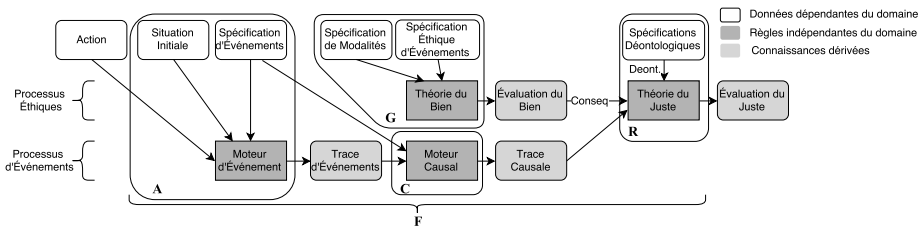

Figure 1. Modèles et modularité. **A**, **C** **G** *et* **R** *représentent respectivement les modèles d'action* $\mathbb{A}$*, de causalité* $\mathbb{C}$*,du Bien* $\mathbb{G}$ *et du Juste* $\mathbb{R}$ *formant le cadre d'évaluation éthique* **F**.

Nous présentons maintenant ces différents modèles, leur composantes et leurs entrées et sorties, illustrées dans la figure 1. Un *modèle d'action* $\mathbb{A}$ permet à l'agent de

représenter son environnement et les changements qui s'y déroulent. Il prend comme entrée un *ensemble de scenarios* $\mathbb{S}$, qui définit l'ensemble d'actions considérées (i.e. envisagées). Il est composé d'une *situation initiale* contenant les fluents vrais à T=0, une *spécification d'évènements* contenant un ensemble d'évènements et de relations de dépendance, et un *moteur d'évènements* qui permet à la simulation d'évoluer. Il génère une *trace d'évènements* de chaque simulation qui désigne pour chaque moment les évènements qui s'y produisent et des fluents qui y sont vrais. Un *modèle de causalité* $\mathbb{C}$ identifie toutes les conséquences des actions, directes ou non, rendant possible un raisonnement sur la responsabilité et l'imputabilité des agents. Il prend comme entrée la *trace d'évènements* produite par le modèle d'action et la *spécification d'évènements* de ce dernier. A partir du *moteur causal*, il génère une *trace causale* de chaque simulation qui identifie les liens de cause à effet qui existent entre les évènements. Un *modèle du Bien* $\mathbb{G}$ donne une appréciation de la valeur éthique intrinsèque de finalités ou d'évènements. Il est composé de quelques spécifications propres (*spécification de modalités* et *spécification éthique d'évènements*) ainsi que d'une ou plusieurs *théories du Bien*. Il génère une *évaluation du Bien*, évaluant les évènements comme plus ou moins en accord avec le Bien. Un *modèle du Juste* $\mathbb{R}$ détermine l'action la plus juste selon des circonstances données. Il prend comme entrée la *trace causale* produite par le modèle de causalité et, si la théorie du Juste considérée contient des principes conséquentialistes, une *évaluation du Bien* produite par le modèle du Bien. Il est composé d'une ou plusieurs *théories du Juste* et, si l'une des théorie du Juste le requiert, d'un *ensemble de spécifications déontologiques*. Il génère une *évaluation du Juste*, évaluant les actions comme plus ou moins en accord avec le Juste, et donc admissibles ou non. Celle-ci est représentée par le prédicat $\texttt{per}(j,\alpha)$ (permissible) où α est l'action considéré et j un label désignant le principe éthique présidant à cette évaluation. Ce label permet de générer des jugements selon plusieurs principes éthiques. Sur la base de ces modèles, le *cadre d'évaluation éthique* est défini par:

$$\mathbb{F} = \langle \mathbb{A}, \mathbb{C}, \mathbb{G}, \mathbb{R} \rangle$$

Étant donné un cadre d'évaluation éthique $\mathbb{F} = \langle \mathbb{A}, \mathbb{C}, \mathbb{G}, \mathbb{R} \rangle$ et un ensemble de scénarios $\mathbb{S}$ correspondant à un ensemble $\mathcal{A}_c$ d'actions considérées (voir section 4.3), nous définissons l'ensemble des actions admissibles selon le principe éthique j comme suit (en utilisant les notations introduites en 2.3):

$$Admissible(\mathbb{F}, j, \mathcal{A}_c) = \{a \in \mathcal{A}_c \mid \mathbb{S} \cup \mathbb{A} \cup \mathbb{C} \cup \mathbb{G} \cup \mathbb{R} \models \texttt{per}(j, a)\}$$

Chaque module du cadre étant un programme logique, il est possible de tous les regrouper (par simple union de leurs règles) en un unique programme logique permettant de dériver l'admissibilité ou non des actions considérées. Si cette solution est réalisable en pratique sur de petits problèmes, la phase d'instantiation des solveurs ASP s'en trouve pénalisée par l'impossibilité de se baser sur les conclusions des modèles précédents pour limiter le nombre d'instantiations des règles des modèles en fin de chaine. C'est la raison pour laquelle nous avons identifié les entrées et sorties de chaque module. En procédant par dérivations successives des conclusions de chaque théorie, on évite l'explosion combinatoire des instantations futures et l'on accélère

le processus total (d'un facteur 8 à 14 sur nos exemples).La compartimentation des différents types de processus permet leur analyse spécifique. Remplacer un modèle spécifique tout en maintenant les autres rend possible l'examen individualisé de ses ramifications. Une telle modularisation de la dérivation des conclusions de chaque théorie rend de plus relativement aisé le remplacement de l'un des modules par une procédure externe. Ainsi, tant qu'il reste possible d'en décoder le résultat selon le format défini pour nos traces, il serait envisageable de remplacer le moteur d'évènements par un planificateur SAT par exemple, ou le moteur causal par un processus bayesien.

4. Modèle d'action

Le modèle d'action est la brique fondamentale de notre cadre, car c'est lui qui indique la façon dont va s'exprimer la dynamique de notre système, indiquant comment les différentes actions ou évènements affectent le monde en fonction de leur contexte d'exécution. En pratique, ce sont les prédicats permettant d'exprimer occurences, effets et préconditions des évènements qui sont le point clef du cadre, car ils se retrouvent à tous les niveaux. Sur la base de ce vocabulaire, le modèle d'action comprend une partie générique, le moteur d'évènement, qui spécifie les règles régissant la dynamique du systèmes et une partie spécifique à chaque domaine ou situation, qui définit les différents évènements du domaine considéré et la situation initiale.

Nous présentons ici un cadre de base fondé sur le calcul des évènements discret, un formalisme adapté à la description de trames narratives qui se traduit bien en ASP et nous parait avoir des avantages de lisibilité. Les conséquences indirectes sont gérées par le déclenchement d'évènements dits « automatiques » pouvant ainsi produire des réactions en chaine. Nous nous plaçons dans un cadre déterministe, aussi bien au niveau des effets des actions qu'au niveau du déclenchement des évènements automatiques. Il est ainsi toujours possible de déterminer sans ambiguité dans quel état mène un ou plusieurs évènements et quels évènements automatiques se déclenchent dans un état donné. Afin de garder des primitives simples pour ne pas alourdir les modèles se basant sur celui-ci, nous décrivons nos évènements à la manière d'actions de type STRIPS, en considérant des effets directs (positifs ou négatifs) et des préconditions simples (conjonctions de fluents positifs ou négatifs).Il est ainsi possible de traduire des problèmes exprimés dans ce fragment du formalisme PDDL utilisé en planification (McDermott *et al*., 1998). Des notions plus complexes (préconditions quantifiées, effets conditionnels ou indirects) peuvent néanmoins se simuler par « aplanissement » de l'ensemble d'action (Fox, Long, 2003). L'utilisation d'évènements automatiques peut aussi permettre de gérer des effets indirects ou conditionnels, comme l'illustre notre exemple de dilemme médical (voir 4.2).

Nous proposons donc un ensemble de prédicats très classiques permettant de spécifier ces différents évènements et d'exprimer le déroulement d'une simulation. D'autres approches, comme par exemple les langages d'action (Gelfond, 2008), sont envisageables, mais un enrichissement de l'expressivité des traces ou des spécifications d'évènements nécessite pour sa prise en compte une adaptation des autres modèles.

(Berreby *et al.*, 2018) présente un enrichissement de ce modèle d'action de base afin d'expliciter dans la trace les actions qu'un agent choisit de ne pas exécuter et la prise en compte de cette modification dans le moteur causal.

4.1. Moteur d'évènement

Le calcul des évènements adapté

Le moteur d'évènement présenté ici correspond au calcul des évènements discret (DEC) décrit dans (Mueller, 2008), avec les ajouts suivants. Pour répondre aux particularités de la modélisation de scénarios éthiques complexes, nous introduisons des *évènements automatiques* en plus des actions. Ces évènements automatiques se produisent lorsque toutes leurs conditions préalables, sous la forme de fluents, tiennent, sans décision de l'agent. Ils correspondent à une systématisation de la notion d'évènements déclenchés, réprésentés usuellement par des règles de déclenchement. Afin d'assurer que ce déclenchement reste déterministe, nous introduisons aussi une notion de priorité entre les évènements qui permet d'éviter certains déclenchements incompatibles ainsi que nous le discutons plus bas au niveau des axiomes de préconditions. Notons que de part ce déterminisme, le module d'action n'introduit pas d'ambiguité entre alternatives différentes et comme il ne contient pas non plus de sources d'incohérence (ni contraintes d'intégrité, ni négation explicite), il produit un unique Answer Set dans notre contexte (pour peu qu'il ait une unique entrée).

Enfin, nous introduisons un ensemble de *simulations* qui permettent à l'agent de simuler séparément et simultanément les effets de différentes actions sur le même contexte initial. Cet ajout est significatif puisqu'il nous fait basculer d'une représentation linéaire du temps caractérique du calcul des évènements avec un unique axe temporel à une représentation sur plusieurs axes qui se rapproche d'une représentation plus arborescente tel qu'utilisée dans le calcul des situations (Schiffel, Thielscher, 2006) ou des notions de moments (correspondant à un temps T) et d'historiques associées à un monde (correspondant à une simulation S) utilisées dans la sémantique de DL-MA (Lorini, 2012). Chaque simulation correspond ainsi à un déroulé possible de la situation selon les actes de l'agent, divisé en plusieurs moments successifs.

Dans la suite de cet article, et en particulier dans les définitions formelles où nous devrons définir des ensembles d'atomes instantiés, nous désignons les différents domaines de valeurs relatifs aux éléments de notre modèle comme suit: $\mathcal{S}$ est l'ensemble des simulations, $\mathcal{T}$ l'ensemble des moments, $\mathcal{F}$ l'ensemble des fluents (positifs ou négatifs), $\mathcal{A}$ l'ensemble des actions, $\mathcal{U}$ l'ensemble des évènements automatiques et $\mathcal{E}$ l'ensemble des évènements où $\mathcal{E} \equiv \mathcal{A} \cup \mathcal{U}$. Pour faciliter la lecture du code fourni et éviter de devoir préciser le domaine de chaque variable lorsque nous introduisons un prédicat, nous nommons nos variables selon la même nomenclature, utilisant des variables `T`, `T1`, `T2` pour les moments, `F`, `F1`, etc. pour les fluents et ainsi de suite. Ces domaines sont précisés factuellement par respectivement les prédicats `time(T)`, `fluent(F)`, `event(E)`, `act(A)` et `auto(U)`.

L'objet principal du moteur d'action est d'inférer à partir de la situation initiale et des actions effectuées l'ensemble des évènements qui se produisent et l'évolution des fluents à chaque étape des différentes simulations, qui sont représentées ainsi :

– `occurs(S,E,T)` indique que l'évènement `E` se produit au temps `T` dans la simulation `S`. Plus précisément cela signifie que, dans `S`, l'évènement `E` se déclenche au temps `T`, menant au temps `T+1` où ses effets auront été appliqués.

– `holds(S,F,T)` indique que le fluent `F` est vérifié au temps `T` dans la simulation `S`. Normalement, `F` est un fluent positif (non précédé de `neg`) et cela signifie donc que `F` est vrai à `T` dans `S`, ce fluent étant considéré comme faux si ce prédicat n'est pas présent. Toutefois, en présence de préconditions négatives, `holds(S,neg(F),T)` est utilisé pour indiquer explicitement que `F` est faux à `T` dans `S`.

Axiomes d'effets

Ces axiomes caractérisent le comportement des fluents qui contribuent à la création d'évènements. Ils ont pour objet de définir de façon unique quels prédicats de type `holds(S,F,T)` sont vérifiés dans les différents états. Pour déterminer ces états, ces axiomes s'appuient sur quelques prédicats. `initially(F)` et `effect(E,F)`, donnés par le contexte d'action (voir 4.2) indiquent respectivement que `F` est vrai initialement et l'évènement `E` a pour effet de causer `F` (qui peut être positif ou négatif). `negative(F)` identifie les fluents négatifs (i.e. les négations de fluents) utilisés dans le contexte. `initiates(S,E,F,T)` et `terminates(S,E,F,T)` indiquent respectivement qu'un évènement `E` survenant au temps `T` dans `S` rend le fluent (positif) `F` vrai ou faux. `clipped(S,F,T)` signifie que dans la simulation `S`, le fluent (positif) `F` est falsifié (rendu faux) par quelque évènement survenant au temps `T`. Ainsi :

```
negative(neg(F)):-effect(E,neg(F)).
initiates(S,E,F,T):-effect(E,F),occurs(S,E,T),not negative(F).
terminates(S,E,F,T):-effect(E,neg(F)),occurs(S,E,T),sim(S),time(T).
clipped(S,F,T):-terminates(S,E,F,T).
```

Cela nous permet d'axiomatiser les principes qui gouvernent les fluents: un fluent est vrai si il vient d'être initié par un évènement (ou l'était initialement si T=0); un fluent précédemment vrai le reste jusqu'à ce qu'un évènement le termine; il est faux dans les autres cas (ce que l'on explicite pour les préconditions négatives). Notons que la troisième règle, qui traduit l'inertie permet, grâce à la negation par l'échec, de gérer le *problème du cadre*. De plus, comme seule la quatrième règle, qui vérifie que l'on ait pas `holds(S,F,T)`, peut dériver un atome de la forme `holds(S,neg(F),T)`, on est assuré de ne pas avoir à la fois `holds(S,F,T)` et `holds(S,neg(F),T)`.

```
holds(S,F,0):-initially(F),sim(S).
holds(S,F,T):-initiates(S,E,F,T-1),time(T).
holds(S,F,T):-holds(S,F,T-1),not clipped(S,F,T-1),time(T).
holds(S,neg(F),T):-prec(neg(F),E),not holds(S,F,T),time(T).
```

Axiomes de préconditions

Ces axiomes caractérisent le comportement et le déclenchement des évènements (qui determinent l'état des fluents). Ils ont pour objet de définir de façon unique quels prédicats de type `occurs(S,E,T)` sont vérifiés. Pour déterminer cela, ces axiomes utilisent différents prédicats. Tout d'abord, à partir du prédicat `prec(F,E)`, donné par le contexte, qui indique que F est une précondition de E, nous définissons deux prédicats pour gérer la satisfaction des préconditions : `incomplete(S,E,T)` indique que `E` est incomplet à `T` dans `S`, c'est-à-dire que toutes ses préconditions ne sont pas remplies; `possible(S,E,T)` indique que `E` est possible à `T` dans `S`, c'est-à-dire que toutes ses préconditions sont remplies.

Comme il est possible que plusieurs évènements incompatibles soient possibles et tentent de se produire en même temps, on distingue le déclenchement d'un évènement, c'est-à-dire le fait que l'évènement « tente » de s'exécuter, de son occurence réelle, c'est-à-dire le fait qu'il se produise effectivement dans la simulation. Ces cas de simultanéité sont gérés par des priorités entre les évènements lorsqu'on souhaite éviter que deux évènements se déclenchent simultanément. Cela se fait dans le contexte à l'aide du prédicat `priority(E2,E1)` qui indique que `E2` est prioritaire sur `E1`. Deux évènements peuvent en effet être incompatibles, typiquement lorsque l'un bloque l'autre (en falsifiant l'une de ses précondition) ou lorsqu'ils produisent des effets opposés. Nous exigeons ici qu'en cas d'incompatibilité de deux évènements susceptibles d'être possibles en même temps, l'un des deux évènements soit univoquement désigné comme prioritaire pour assurer un déterminisme du déclenchement. Il serait possible de permettre un non-déterminisme dans l'ordre de déclenchement entre deux évènements E1 et E2 en indiquant qu'ils tous deux prioritaire l'un sur l'autre, ce qui provoquerait la génération de plusieurs Answer Sets selon les différentes ordres possibles, mais nous laissons ce genre de cas à des travaux futurs. Si deux évènements sont compatibles, en revanche, il est possible de ne définir aucune priorité entre eux pour permettre leur occurence simultanée. On gère ainsi les priorités avec les prédicats suivants : `triggered(S,E,T)` indique que E est déclenché à T dans S; `overtaken(S,E,T)` indique que E ne peut se produire à T dans S à cause de l'occurence d'un autre évènement qui lui est prioritaire. De plus, nous utilisons le prédicat `performs(S,A,T)`, indiquant que la décision de l'agent d'effectuer `A` à `T` dans `S`, qui sera donné comme entrée dans les scénarios (voir 4.3).

Ces prédicats nous permettent d'axiomatiser les principes qui régissent les évènements: un évènement automatique se déclenche quand toutes ses préconditions sont vraies; une action se déclenche si toutes ses préconditions sont vraies et qu'un agent l'effectue; tout évènement déclenché se produit à moins qu'il ne soit invalidé par l'occurence d'un autre évènement qui lui soit prioritaire.

```
incomplete(S,E,T):-prec(F,E),not holds(S,F,T),sim(S),time(T).
possible(S,E,T):-not incomplete(S,E,T),sim(S),event(E),time(T).
triggered(S,U,T):-possible(S,U,T),auto(U).
triggered(S,A,T):-possible(S,A,T),performs(S,A,T),act(A).
overtaken(S,E1,T):-triggered(S,E1,T),occurs(S,E2,T),priority(E2,E1).
occurs(S,E,T):-triggered(S,E,T),not overtaken(S,E,T).
```

4.2. Contexte d'action

Le *contexte d'action* est composé d'une *spécification des évènements* dépendante du domaine et d'une *situation initiale*. La spécification des évènements définit les évènements existants, leur préconditions et effets (`prec(F,E)` et `effect(E,F)`), ainsi que les éventuelles priorités entre eux (`priority(E1,E2)`).La *situation initiale* définit les fluents qui sont initialement vrais, par un ensemble de faits de la forme `initially(F)`. Elle permet en outre de préciser le vocabulaire en définissant les objets présents. Ces deux éléments du contexte correspondent en pratique aux spécifications de domaines et de problèmes en PDDL (McDermott *et al.*, 1998).

C'est l'union du moteur d'évènements (ensemble des axiomes d'effets et de préconditions) et d'un contexte d'action Ctx qui forme le modèle d'action, noté ici $\mathbb{A}_{Ctx}$. Nous proposons deux exemples qui seront utilisés tout au long de cet article pour illustrer les aspects spécifiques au contexte considéré.

Exemple 1: un dilemme médical

Un médecin (l'agent autonome) est en possession de trois différents traitements expérimentaux pour une maladie grave et handicapante. Chaque traitement a des effets différents selon la variante de la maladie qui atteint le patient (parmi 6 variantes possibles). Il n'existe aucun test pour déterminer a priori quel patient a quelle variante et les effets sont trop rapides pour que le médecin ne puisse ajuster son traitement à posteriori, il ne peut donc individualiser le traitement et doit décider du traitement qu'il donne à l'ensemble de ses patients (on considère ici 100 patients). Toutefois, il dispose de données sur la répartition de chaque variante dans la population globale qui lui permettent d'estimer le nombre de patients ayant chaque variante parmi les 100 considérés. Ainsi la répartition selon les 6 variantes notées g_1 à g_6 est la suivante : 15 %, 20 %, 25 %, 25 %, 10 %, 5 %. Le médecin sait de plus que

– le traitement Alpha peut soigner la variante g_1 mais provoque la mort des patients atteints de la variante g_2. Ainsi, pour 100 patients qui essaient le traitement Alpha, 15 sont guéris, 20 perdent leur vie et 65 restent inchangés.

– le traitement Beta peut soigner les variantes g_3 et g_6 mais provoque la mort des patients atteints de la variante g_4. Ainsi, pour 100 patients qui essaient le traitement Beta, 30 sont guéris, 25 perdent leur vie et 45 restent inchangés.

– le traitement Gamma peut soigner la variante g_2 mais provoque la mort des patients atteints des variantes g_1, g_5 et g_6. Cependant, ce traitement permet de sauver des patients affectés par les autres variantes en transplantant le foie d'un patient dont il a causé la mort. Ainsi, pour 100 patients qui essaient le traitement Gamma, 50 sont guéris (20 directement et 30 grâce à une transplantation issue des morts), 30 perdent leur vie et 20 restent inchangés.

Pour modéliser cet exemple, on divise les 100 patients en 20 groupes de 5 personnes auxquels on attribue les différentes variantes de la maladie (vu que les proportions données sont toutes divisibles par 5). Chaque traitement peut avoir, pour chaque variante de la maladie, des effets positifs (`posImpact`) ou négatifs (`negImpact`). Notons que si un traitement ne peut avoir à la fois des effets positifs et négatifs pour la même variante, il peut n'avoir aucun effet sur une variante. On indique qu'initialement tout les groupes sont vivants (`alive`) et malades (`sick`), en indiquant par quelle va-

riante est touchée chaque groupe (`type(`X`,`V`)` signifie que le groupe X est touché par la variante V). La partie *situation initiale* et *déclarations des constantes du domaine* qui en résulte est donné ci dessous.

```
% ------------ DOMAINES ---------------------------------------- %
time(0..3).                          treatment(alpha;beta;gamma).
group(1..20).                        variant(g1;g2;g3;g4;g5;g6).
numberInGroup(X,5):-group(X).
posImpact(alpha,g1).                 negImpact(alpha,g2).
posImpact(beta,(g3;g6)).             negImpact(beta,g4).
posImpact(gamma,g2).                 negImpact(gamma,(g1;g5;g6)).
% ------------ SITUATION INITIALE------------------------------- %
initially(type(1..3,g1)).            initially(type(4..7,g2)).
initially(type(8..12,g3)).           initially(type(13,g6)).
initially(type(14..18,g4)).          initially(type(19..20,g5)).
initially(sick(X)):-group(X).        initially(alive(X)):-group(X).
```

Nous définissons de plus l'action `give(Z)` où `Z` est un traitement, indiquant le choix du médecin. Les réactions des patients au traitement sont données par les évènements automatiques `cures(X,G,Z)` et `kills(X,G,Z)` où `X` est un groupe, `G` une variante, et `Z` un traitement. Enfin, nous définissons `transplant(X1,X2)` comme un évènement automatique en considérant que s'il choisit le traitement Gamma, le médecin fait nécessairement toutes les transplantations qu'il peut. Le fait de considérer comme évènement automatique ce qui pourrait relever d'une décision revient ici à inclure cet aspect décisionnel dans l'action `give(gamma)`. Cette action correspond donc plus à une décision de « suivre le protocole de traitement Gamma », sans possibilité pour le médecin d'altérer le protocole.

Cet évènement de transplantation est un exemple d'évènement dont plusieurs instantiations peuvent être incompatibles entre elles. Il n'est en effet pas possible pour quelqu'un de donner son foie à deux personnes différentes (`transplant(X1,X2)` et `transplant(X1,X3)` incompatibles pour `X2` différent de `X3`), ni pour quelqu'un de recevoir deux transplantations de foies simultanées (`transplant(X1,X2)` et `transplant(X3,X2)` incompatibles pour `X1` différent de `X3`). Il faut donc déclarer des priorités entre ces évènements incompatibles. Nous privilégions arbitrairement les transplantations impliquant le patient de numéro le plus faible.

Notons enfin que cette décomposition de traitements complexes en une décision déclenchant un certain nombre d'évènements automatiques illustre aussi une façon de modéliser des effets universels et/ou conditionnels : la décision d'appliquer un traitement s'applique à tous les patients et, selon la variante de la maladie qui les affecte, peut les soigner ou les tuer (ce qui peut alors permettre une transplantation). Nous obtenons la spécification d'évènements suivante :

```
% ------------ SPECIFICATION D'EVENEMENTS----------------------- %
event(E):-auto(E).                   event(E):-act(E).
% - - - Action : give(Z)
act(give(Z)):-treatment(Z).
prec(sick(X),give(Z)):-group(X),act(give(Z)).
effect(give(Z),received(X,Z)):-group(X),act(give(Z)).
% - - - Event automatique  : cures(X,G,Z)
auto(cures(X,G,Z)):-treatment(Z),group(X),variant(G),posImpact(Z,G).
```

```
prec(received(X,Z),cures(X,G,Z)):-auto(cures(X,G,Z)).
prec(alive(X),cures(X,G,Z)):-auto(cures(X,G,Z)).
prec(sick(X),cures(X,G,Z)):-auto(cures(X,G,Z)).
prec(type(X,G),cures(X,G,Z)):-auto(cures(X,G,Z)).
effect(cures(X,G,Z),neg(sick(X))):-auto(cures(X,G,Z)).
% - - - Event automatique  : kills(X,G,Z)
auto(kills(X,G,Z)):-treatment(Z),group(X),variant(G),negImpact(Z,G).
prec(received(X,Z),kills(X,G,Z)):-auto(kills(X,G,Z)).
prec(alive(X),kills(X,G,Z)):-auto(kills(X,G,Z)).
prec(type(X,G),kills(X,G,Z)):-auto(kills(X,G,Z)).
effect(kills(X,G,Z),dead(X)):-auto(kills(X,G,Z)).
effect(kills(X,G,Z),neg(alive(X))) :-auto(kills(X,G,Z)).
% - - - Event automatique  : transplant(X1,X2)
auto(transplant(X1,X2)):-group(X1;X2).
prec(dead(X1),transplant(X1,X2)):-auto(transplant(X1,X2)).
prec(received(X1,gamma),transplant(X1,X2)):-auto(transplant(X1,X2)).
prec(sick(X2),transplant(X1,X2)):-auto(transplant(X1,X2)).
prec(received(X2,gamma),transplant(X1,X2)):-auto(transplant(X1,X2)).
prec(alive(X2),transplant(X1,X2)):-auto(transplant(X1,X2)).
effect(transplant(X1,X2),neg(sick(X2))):-auto(transplant(X1,X2)).
effect(transplant(X1,X2),neg(received(X1,gamma))):-auto(transplant(X1,X2)).
% - - - Définition des priorités
priority(transplant(Y,X1),transplant(Y,X2)):-
            X1<X2,auto(transplant(X1,Y);transplant(X2,Y)).
priority(transplant(Y1,X),transplant(Y2,X)):-
            Y1<Y2,auto(transplant(X,Y1);transplant(X,Y2)).
```

Exemple 2 : problème du trolley

Le problème du trolley est un exemple canonique de dilemme éthique (Singer, 2005), souvent utilisé pour illustrer la doctrine du double effet, par comparaison des variantes dites *(switch)* et *(push)*. Nous modélisons dans un même exemple ces deux options, en changeant le nombre de personnes sur la voie secondaire à deux personnes (plutôt qu'une) pour mieux différencier les résultats des différents principes éthiques que nous verrons plus loin. La situation est donc la suivante :

Un train roule vers cinq personnes travaillant sur les voies. Si l'agent ne fait rien, le train les écrasera, les tuants tous les 5. Toutefois, l'agent peut actionner un levier d'aiguillage (switch), qui déviera le train de sa voie actuelle vers une voie secondaire sur laquelle se trouve deux autres personnes, mais dans ce cas, le train écrasera ces personnes, les tuant. De plus, il y a un pont au dessus de la voie sur lequel se tient une personne. L'agent peut donc aussi le pousser sur les rails (push), auquel cas, le train le renversera, le tuant, mais sera stoppé par cette collision et n'écrasera ainsi pas le groupe de cinq personnes.

Nous pouvons alors modéliser cette situation avec le contexte suivant.

```
% ------------ DOMAINES ----------------------------------------- %
time(0..6).    track(side(0..4);main(0..4)).
buttonOn(main(0)).                bridgeOn(b,main(1)).
object(g1;g2;g3).                 group(g1;g2;g3).
numberInGroup(g1,5;g2,2;g3,1).
% ------------ SITUATION INITIALE------------------------------- %
initially(on(g1,main(3))).        initially(on(g2,side(2))).
initially(on(g3,b)).              initially(on(train,main(0))).
initially(alive(g1;g2;g3)).
% ------------ SPECIFICATION D'EVENEMENTS----------------------- %
event(E):-auto(E).                event(E):-act(E).
% - - - Action : SWITCH
act(switch(M)):-buttonOn(M).
```

```
prec(on(train,M),switch(M)):-act(switch(M)).
effect(switch(M),neg(on(train,M))):-act(switch(M)).
effect(switch(main(N)),on(train,side(N))):-act(switch(main(N))).
effect(switch(side(N)),on(train,main(N))):-act(switch(side(N))).
% - - - Action : PUSH
act(push(O,B)):-object(O),bridgeOn(B,_),initially(on(O,B)).
prec(on(O,B),push(O,B)):-act(push(O,B)).
effect(push(O,B),neg(on(O,B))):-act(push(O,B)).
effect(push(O,B),on(O,M)):-act(push(O,B)),bridgeOn(B,M).
% - - - Event automatique  : run
auto(run(train,M)):-track(M).
prec(on(train,M),run(train,M)):-auto(run(train,M)).
effect(run(train,main(N-1)),on(train,main(N))):-auto(run(train,main(N))).
effect(run(train,side(N-1)),on(train,side(N))):-auto(run(train,side(N))).
effect(run(train,M),neg(on(train,M))):-auto(run(train,M)).
% - - - Event automatique  : crash
auto(crash(G,M)):-object(G),track(M).
prec(on(G,M),crash(G,M)):-auto(crash(G,M)).
prec(on(train,M),crash(G,M)):-auto(crash(G,M)).
effect(crash(G,M),neg(alive(G))):-auto(crash(G,M)).
effect(crash(G,M),neg(on(train,M))):-auto(crash(G,M)).
% - - - Définition des priorités
priority(crash(G,M),run(train,M)):-auto(crash(G,M)).
priority(A,run(train,M)):-act(A),track(M).
```

4.3. Scénarios

Maintenant que nous avons défini et illustré les composants du modèle d'action, nous nous intéressons à ses entrées et sorties. Comme il l'a été dit plus haut, le cadre d'évaluation éthique prend comme entrée un ensemble de scénarios $\mathbb{S}$ correspondant à un ensemble d'actions considérées $\mathcal{A}_c$. Nous définissons ci-dessous la notion de scénario.

DÉFINITION 1. — *Etant donné un contexte d'action Ctx, un scénario de Ctx est défini comme un couple (s, P) tel que $s \in \mathcal{S}$ est une simulation et $P \subseteq$* `performs(`$s, \mathcal{A}_c, \mathcal{T}$`)` *est un* ensemble d'actions accomplies *dans s tel que toute action accomplie se produit effectivement, soit* $\forall$`performs(s,a,t)`$\in P$, $\mathbb{A}_{Ctx} \cup P \models$`occurs(s,a,t)`.

La définition d'un scénario exige toutes les actions accomplies dans P se produisent effectivement dans s au temps indiqué selon les règles du moteur d'évènement. Cela suppose que l'on n'inclut dans P que des actions possibles au temps indiqué dans s (toutes les préconditions doivent être vérifiées) et non invalidées par des évènements prioritaires. Bien que cette définition soit plus générale, dans le cadre de cet article, nous nous concentrerons uniquement sur les cas où P est un singleton intervenant au temps 0 (ie $P = \{$`performs(`$s, a, 0$`)`$\}$ où $a \in \mathcal{A}_c$). L'ensemble $\mathbb{S}$ qui sert d'entrée à notre cadre peut ainsi s'exprimer comme un ensemble de faits définissant les différentes simulations et l'action accomplie dans chacune d'entre elles (ensemble de faits de la forme `sim(S)` ou `performs(S,A,T)`). Nous illustrons cela sur nos deux exemples.

Exemple 1& 2 (2) : scénarios

Pour l'exemple 1, le médecin envisage trois scénarios selon le traitement choisi. L'ensemble des actions considérées est donc $\mathcal{A}_c = \{\texttt{give(alpha)}, \texttt{give(beta)}, \texttt{give(gamma)}\}$. Il nous faut donc trois simulations ($\mathcal{S} = \{\texttt{s0}, \texttt{s1}, \texttt{s2}\}$) qui correspondent à trois scénarios de la forme $\sigma_i = (\texttt{si}, \{\texttt{performs(si,give(}\nu_i\texttt{),0}\})$ où $\nu_1 = \texttt{alpha}$, $\nu_2 = \texttt{beta}$ et $\nu_3 = \texttt{gamma}$. $\mathbb{S}$ s'exprime donc ainsi :

```
sim(s0;s1;s2).                  performs(s0,give(alpha),0).
performs(s1,give(beta),0).      performs(s2,give(gamma),0).
```

Dans le cas du trolley, `s0` correspond à `push(g3,b)` et `s1` à `switch(main(0))`.

4.4. Trace d'évènements

Etant donné un ensemble de scénarios Σ en entrée, le modèle d'action peut alors dériver pour chaque scénario l'évolution des fluents et les occurences d'évènements qui en découlent. La trace d'exécution d'un scénario peut alors se définir ainsi:

DÉFINITION 2. — *Etant donné un contexte d'action Ctx, la* trace d'exécution *d'un scénario $\sigma = (s, P)$ est définie comme $tr_{Ctx}(\sigma) = Th_{\mathcal{L}_{h,o}(s)}(\mathbb{A}_{Ctx} \cup P)$ où* $\mathcal{L}_{h,o}(s) = \texttt{holds}(s, \mathcal{F}, \mathcal{T}) \cup \texttt{occurs}(s, \mathcal{E}, \mathcal{T})$.

La *trace d'évènements* d'un ensemble de scénarios (représenté sous forme de faits dans $\mathbb{S}$) est alors l'union des traces d'exécutions de chacun de ces scénarios. Elle est notée $tr_{Ctx}(\mathbb{S})$. Nous avons alors $tr_{Ctx}(\mathbb{S}) = Th_{\mathcal{L}_{h,o}(\mathcal{S})}(\mathbb{A}_{Ctx} \cup \mathbb{S})$. Comme nous le verrons par la suite, cette trace, accompagnée de la spécification d'évènements, peut servir d'entrée au modèle causal.

Génération des scénarios

Nous supposons dans cet article une spécification factuelle des différents scénarios considérés, mais il est assez direct en ASP de générer automatiquement tous les scénarios possibles en utilisant des axiomes de choix. Par exemple, pour notre example médical, on peut utiliser `1{performs(s0,give(Z),0):treatment(Z)}1` qui signifie qu'on doit choisir pour chaque Answer Set un et un seul Z tel que Z est un traitement. Cependant, chaque scénario correspond alors à un Answer Set différent contenant la seule simulation `s0` avec différentes actions accomplies selon l'Answer Set, et les autres modules ne peuvent comparer les simulations entre elles. Ce n'est toutefois pas un problème si l'on dérive modulairement les traces, chaque Answer Set permettant d'obtenir la trace d'un scénario (mais en utilisant systématiquement le même nom pour la simulation, ici s_0). Pour obtenir la trace de tous les scénarios, il suffit alors renommer différemment dans chaque Answer Set la simulation s_0 et de prendre l'union de toutes les traces obtenues comme entrée des modules la réclamant.

Nous illustrons pour finir la notion de trace en donnant des extraits de la trace de nos deux exemples. La trace complète peut s'obtenir à partir du code fournis.

Exemple 1 (3) : trace d'évènements du dilemme médical

La trace obtenue pour l'exemple médical, triée par simulation et temps pour plus de lisibilité, est la suivante (en omettant les holds pour s_1 et s_2, et en rassemblant les faits avec les notations « tt .. » et « tt ; »):

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  Sim S0  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Temps 0 : Holds initiaux omis (fluents 'type', 'sick' et 'alive')
occurs(s0,give(alpha),0).
% Temps 1 :fluents 'type', 'sick' et 'alive' : idem temps 0, omis ici
holds(s0,received(1..20,alpha),1).
occurs(s0,cures(1..3,g1,alpha),1).   occurs(s0,kills(4..7,g2,alpha),1).
% Temps 2+ : fluents 'type', 'received' : idem temps 1, omis ici
holds(s0,sick(4..20),2).             holds(s0,dead(4..7),2).
holds(s0,alive(1..3),2).             holds(s0,alive(8..20),2).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  Sim S1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
occurs(s1,give(beta),0).
occurs(s1,cures(8..12,g3,beta),1).   occurs(s1,cures(13,g6,beta),1).
occurs(s1,kills(14..18,g4,beta),1).
holds... % sick : 1-7,14-20, dead : 14-18, alive : 1-13,19-20
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  Sim S2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
occurs(s2,give(gamma),0).
occurs(s2,cures(4..7,g2,gamma),1).   occurs(s2,kills(1..3,g1,gamma),1).
occurs(s2,kills(19..20,g5,gamma),1). occurs(s2,kills(13,g6,gamma),1).
occurs(s2,transplant((1,8);(2,9);(3,10);(13,11);(19,12);(20,14)),2).
holds... % sick : 1-3,13,15-20, dead : 1-3,13,19-20, alive : 4-12,14-18
```

Exemple 2 (3) : trace pour le problème du trolley

Pour le problème du trolley, on omet les `holds` pour plus de lisibilité. La trace obtenue est alors la suivante:

```
%--  push  --%
occurs(s0,push(g3,b),0).             occurs(s0,run(train,main(0)),1).
occurs(s0,crash(g3,main(1)),2).
%-- switch --%
occurs(s1,switch(main(0)),0).        occurs(s1,run(train,side(0)),1).
occurs(s1,run(train,side(1)),2).     occurs(s1,crash(g2,side(2)),3).
```

5. Modèle causal

Deuxième bloc important du cadre présenté, le modèle causal s'approche déjà plus de considérations éthiques, la notion de causalité étant essentielle pour peser les conséquences d'un choix donné ou attribuer une part de responsabilité à quelqu'un. Après une brève discussion de cette notion complexe, nous proposons un modèle basique de causalité basé sur notre modèle d'action, que nous étendons ensuite pour raisonner sur ce qu'une action permet d'éviter avant de définir la notion de trace causale.

5.1. Notions de causalité

La causalité est une notion subtile qui a été et reste amplement discutée en philosophie (Hume, 2012 ; J. Halpern, Hitchcock, 2010 ; Pearl, 2003 ; Sosa, Tooley, 1993). Au delà de la philosophie, la définition de cette notion revêt une grande importance

pour d'autres domaines tels que le droit, où elle est centrale à l'attribution de la responsabilité légale (Beebee *et al*., 2009). Dans ce contexte, un agent est typiquement tenu responsable non seulement de ses actions, mais aussi de certains évènements ou changements dans l'état du monde. La question est alors de déterminer quels sont ces évènements et pourquoi ils impliquent la responsabilité de l'agent. Comme le pouvoir causal des actions d'un agent semble constituer le seul lien entre lui et le monde, il est naturel de suggérer que les agents sont responsables des états du monde qu'ils ont causé (Feinberg, 1970) et donc des évènements survenus du fait de ces états. Mais il arrive aussi que l'on tienne responsable un agent pour quelque chose qu'il n'a pas causé, par exemple s'il choisit de ne pas sauver un enfant qui se noie. Ici, le fait qu'il ait eu le pouvoir de changer le resultat implique aussi la notion de causalité. Les discussions sur cette notion ont fait émerger de nombreuses nuances et l'on distingue ainsi, en fonction des approches, différents types de causalité. Une première distinction se fait entre une causalité générale (*type causality*), par exemple "Rouler vite provoque des accidents", et une causalité effective (*actual causality*), par exemple "Le fait que Caitlyn ait roulé vite a provoqué son accident d'aujourd'hui" (J. Y. Halpern, 2015). C'est cette seconde notion qui nous intéresse pour notre modélisation. Une autre distinction est aussi parfois faite au sein de cette causalité effective entre ce qui devrait être considéré comme la cause réelle et ce qui ne devrait être vu que comme circonstanciel (Sloman *et al*., 2009). Dans le cas de l'accident de Caitlyn, on pourrait se demander si c'est sa conduite, la puissance du moteur ou l'existence de la route qui est la cause de l'accident. Cela demande d'évaluer la saillance de chacun des critères pour choisir une cause première. A l'inverse, nous faisons ici le choix de considérer que dans la mesure où tous ces éléments participent d'une façon ou d'une autre au résultat final, ils peuvent tous être considérés comme des causes. En outre, nous nous concentrons ici sur les relations de causalité entre évènements, plutôt qu'entre états, tout changement d'état pouvant être associé à un évènement.

Suivant Hume, une approche courante est de définir la causalité en termes de dépendance contrefactuelle : α *est une cause de* β *quand, si* α *n'était pas arrivé,* β *ne serait pas arrivé non plus*. Cette définition échoue cependant à capturer certaines subtilité de la causalité comme les cas de préemption (une cause pourrait être remplacée par une autre) ou de sur-détermination (il y a plus de causes que nécessaire pour provoquer l'effet). Cele peut s'illustrer en considérant l'exemple suivant : *Suzy lance une pierre sur une bouteille (s-throws) et la brise (shatters). Billy se trouve à côté d'elle avec une seconde pierre. Si Suzy n'avait pas lancé sa pierre, Billy aurait tout de même brisé la bouteille en lançant la sienne (b-throws)*(Kment, 2014). Dans ce cas, *s-throws* n'a pas de dépendance contrefactuelle avec *shatters*, et ne serait donc pas vu comme cause, ce qui est problématique dans un cadre de responsabilité morale. Une autre approche consiste à considérer des modèles causaux structuraux (Pearl, 2003), qui sont très efficaces pour évaluer des dépendances causales entre variables. Cependant, ces modèles prennent mal en compte la dynamique des situations et ne peuvent distinguer entre conditions et transitions ou entre actions et omissions qui sont pourtant des notions clefs dans l'étude de la responsabilité. C'est la raison pour laquelle, suivant les arguments de Hopkins et Pearl en faveur du calcul de situations (Hopkins, Pearl,

2007), nous considérons en premier lieu dans ce cadre une approche fondée sur les modèles d'actions opérant une distinction nette entre états (fluents) et transitions (évènements). Cela n'est toutefois pas incompatible avec l'introduction de considérations contrefactuelles pour nuancer certaines relations. D'autres moteurs causaux prenant cela en compte sont ainsi proposés dans (Berreby *et al*., 2018).

5.2. Un moteur de causalité évènementielle

Axiomes de causalité

En se basant sur l'architecture du calcul d'évènements, nous définissons la causalité en termes de conséquences en s'appuyant sur les effets et préconditions des évènements. comme indiqué nous nous intéressons à des causalités *effectives*, nous nous concentrons sur les évènements qui se produisent effectivement. Ainsi, un fluent F est une *conséquence directe* d'un évènement E survenu au temps T dans S si F est un effet de E qui se réalise effectivement (F est vrai à T+1 dans S). Un évènement automatique U survenu au temps T dans S est une *conséquence directe* d'un fluent F si F est une précondition de U qui permet bien son déclenchement (F est vérifiée au temps T dans S et U se produit bien à ce moment). Enfin, un évènement E2 est une *conséquence* d'un autre évènement E1 survenu au temps T dans S si E2 est une conséquence directe d'un fluent qui est une conséquence directe de E1 ou si E2 est une conséquence d'un évènement E3 qui est lui-même une conséquence de E1.

Pour modéliser cela, nous définissons le prédicat `r(S,causes,E1,T,E2)`, qui indique que l'évènement `E2` est une conséquence de l'évènement `E1` survenu à `T` dans `S`. Une chaîne causale est composée d'une série de fluents et d'évènements, mais le début et la fin d'une chaîne causale sont toujours des évènements. Toutefois les deux premières règles, correspondant aux cas de conséquences directes, utilisent des domaines différents pour le prédicat `r(S,causes,_,T,_)` en substituant un fluent à l'un des évènements. La troisième règle fait de même pour regrouper les deux cas de transitivé, selon que C est un fluent (C=F et Ti correspond au moment où se produit E2) et ou un évènement (C=E3 et Ti correspond alors au moment où se produit E3). Nous considérons de plus qu'une action se cause elle-même afin de faciliter pour la suite l'agrégation des conséquences directes et indirectes d'une action. L'ensemble de règles ainsi obtenu constitue le *moteur causal basique*, noté $\mathbb{C}_b$.

```
r(S,causes,E,T,F):-occurs(S,E,T),effect(E,F),holds(S,F,T+1).
r(S,causes,F,T,U):-holds(S,F,T),prec(F,U),occurs(S,U,T),auto(U).
r(S,causes,E1,T1,E2):-r(S,causes,E1,T1,C),r(S,causes,C,Ti,E2),
     event(E1),event(E2),Ti>T1.
r(S,causes,A,T,A) :- occurs(S,A,T), act(A).
```

5.3. Un moteur de causalité évènementielle avec prévention

Si la responsabilité morale est typiquement associée à l'*occurence* d'évènements, comme la mort ou la guérison des patients traités, la responsabilité est parfois également l'affaire d'éviter des mauvais évènements. C'est en particulier le cas dans notre

exemple du Trolley, où sauver les 5 travailleurs revient en pratique à éviter que ne survienne l'accident qui les tuerait. Il s'agit donc de définir ce que signifie *empêcher* qu'un évènement ne se produise. Il est pour cela nécessaire de s'interesser aux évènements qui auraient pu arriver, et considérer non seulement les évènements directement empêchés par la suppression d'une de leurs conditions nécessaires, mais aussi les évènements qu'ils auraient eux même pu causer. Pour éviter toutefois de prétendre empêcher des évènements qui n'auraient en fait pas pu se produire, il faut toutefois s'assurer que les préconditions non empêchées soit bien toutes remplies à un moment donné -de même que pour gérer des cas de sur-détermination où empêcher une cause ne serait pas suffisant, nous exigeons que l'évènement soit effectivement évité. Nous définissons ci-dessous cette notion, proposant un raffinement de la définition donnée dans (Berreby *et al*., 2015) en précisant la temporalité de la deuxième condition.

DÉFINITION 3. — *Un évènement E_1 (se produisant au temps T dans S)* empêche *un évènement E_2 si:*

*(a) E_1 falsifie (directement ou non) un fluent F qui est une précondition de E_2, en considérant qu'un évènement falsifie indirectement F si F est une conséquence possible d'un fluent F_2 que E_1 falsifie directement (*i.e. *que E_1 rend faux).*

(b) il existe au moins un moment T' postérieur à T dans S durant lequel toutes les préconditions de E_2 non falsifiées par E_1 (directement ou non) sont vraies,

(c) E_2 ne se produit pas après T (dans S).

Pour modéliser cela, on utilise les prédicats suivants: `hypCons(F1,F2)` indique que F2 est une conséquence possible de F1 (pertinente en ce qu'elle intervient en tant précondition d'au moins un évènement); `transTerm(S,E,F,T)` indique que E (qui se produit au temps T dans la simulation S) falsifie directement ou indirectement F (qui doit être un fluent intervenant comme précondition d'au moins un évènement); `r(S,prevents,E1,T,E2)` indique que l'évènement E2 est empêché par l'évènement E1 qui s'est produit à T dans S. Les règles suivantes permettent alors de traduire la notion d'empêchement, en définissant d'abord la notion de falsification directe (deux premières règles définissant `transTerm` selon que l'on s'intéresse à une précondition positive ou négative) ou indirecte (troisième règle). Dans la règle permettant de dériver la relation d'empêchement, les 3 lignes du corps correspondent respectivement aux conditions (a), (b) et (c). `prec(F2,E2):not holds(S,F2,T2), not transTerm(S,E1,F2,T)0` signifie en effet qu'au temps `T2` il n'y a aucune précondition de `E2` qui ne soit ni vraie, ni falsifiée par `E1` et `occurs(S,E2,T3):time(T3),T3>=T0` indique qu'il n'y a dans `S` aucun temps `T3` postérieur à `T` où `E2` se produise. Nous rajoutons de plus une règle de transitivité avec les causes. L'union de $\mathbb{C}_b$ avec l'ensemble de règles ainsi obtenu est nommé le *moteur causal avec prévention* et noté $\mathbb{C}_p$.

```
hypCons(F1,F2):-prec(F1,U),effect(U,F2),prec(F2,E),auto(U).
transTerm(S,E,F,T):-occurs(S,E,T),effect(E,neg(F)),prec(F,E2).
transTerm(S,E,neg(F),T):-occurs(S,E,T),effect(E,F),prec(neg(F),E2).
transTerm(S,E,F2,T):-
     transTerm(S,E,F1,T),hypCons(F1,F2),prec(F2,E2).
```

```
r(S,prevents,E1,T,E2):-
     transTerm(S,E1,F1,T),prec(F1,E2),event(E2),time(T2),
     {prec(F2,E2):not holds(S,F2,T2),not transTerm(S,E1,F2,T)}0,T2>T,
     {occurs(S,E2,T3):time(T3),T3>=T}0.
r(S,prevents,E1,T1,E2):-r(S,causes,E1,T1,E3),r(S,prevents,E3,T2,E2),
     event(E1),event(E2),event(E3),T2>T1.
```

5.4. Trace causale

De même que précédemment pour la trace d'évènements, nous définissons maintenant une trace causale de la façon suivante.

DÉFINITION 4. — *Etant donné un* contexte Ctx, *la* trace causale *d'un scenario* $\sigma = (s, P)$ *(en fonction du moteur causal* $\mathbb{C}_i$*) est définie comme*

$$ctr_{Ctx,i}(\sigma) = Th_{\mathcal{L}_r}(\mathbb{A}_{Ctx} \cup \mathbb{C}_i \cup P)$$

où $\mathcal{L}_r = r(s, \mathcal{R}_i, \mathcal{E}, \mathcal{T}, \mathcal{E})$, $\mathcal{R}_i$ *étant l'ensemble des types de relations causales gérées par le moteur causal. Ici,* $\mathcal{R}_b = \{\texttt{causes}\}$ *et* $\mathcal{R}_p = \{\texttt{causes,prevents}\}$.

En pratique, si on dispose de la trace d'évènements, il est possible de calculer la trace causale de tous les scénarios sans recourir au moteur d'évènements en projetant sur $\mathcal{L}_r$ les conséquences du programme $Ctx \cup tr_{Ctx}(\mathbb{S}) \cup \mathbb{C}_i \cup \mathbb{S}$.

Exemple 1 (4) : trace causale pour le dilemme médical

Dans le cas du dilemme médical, nous utilisons seulement le moteur causal basique. Nous donnons ici un extrait (en regroupant les résultats avec « `..` ») pour montrer que l'on retrouve bien les guérisons et morts causées par le traitement Alpha : `give(alpha)`, survenu au temps 0 dans `s0`, cause `cures(`k`,g1,alpha)` pour $k \in \{1, 2, 3\}$ et `kills(`k`,g2,alpha)` pour $k \in \{4, \ldots, 7\}$. La guérison du groupe 1 atteint de la variante g_1 que soigne Alpha est bien identifiée comme une conséquence de l'administration de Alpha (en pratique `give(alpha)` cause directement `received(1,alpha)` qui cause directement `cures(1,g1,alpha)`). De même, on peut identifier, dans le cas du traitement par Gamma, que la transplantation qui sauve par exemple le groupe 8 (`transplant(1,8)`) est causée par l'administration de Gamma `give(gamma)` (indirectement, à travers la mort du groupe 1). Dans le détail, `give(gamma)` cause `kills(1,g1,gamma)` (le groupe 1 est atteint de la variante g_1 pour laquelle Gamma a un impact négatif) et `kills(1,g1,gamma)` cause `transplant(1,8)` (via la précondition `dead(1)`).

```
r(s0,causes,give(alpha),0,cures(1..3,g1,alpha)).
r(s0,causes,give(alpha),0,kills(4..7,g2,alpha)).
...
r(s2,causes,give(gamma),0,transplant(1,8)).
r(s2,causes,give(gamma),0,kills(1,g1,gamma)).
r(s2,causes,kills(1,g1,gamma),1,transplant(1,8)). [...]
```

Exemple 2 (4) : trace causale pour le problème du trolley

Ici, en revanche, il est nécessaire d'utiliser le moteur causal avec prevention pour pouvoir constater que les deux actions sauvent le groupe de 5 personnes `g1` en empêchant l'accident sur la voie `main(3)`. On a en effet aussi bien `push(g3,b)` que `switch(main(0))` qui empêchent chacun, dans respectivement `s0` et `s1`, l'évènement `crash(g1,main(3))` . On constate que dans le cas du `push`, le fait de pousser n'empêche pas cela par lui-même, mais par transitivité, causant un accident avec `g3` (`%A`) qui, lui, empêche l'accident avec `g1` (`%B`). Le `switch` empêche cette action en empêchant `run(train,main(0))` et donc aussi ses conséquences potentielles comme `crash(g1,main(3))`.

L'extrait de trace causale illustre en effet la multiplicité des conséquences d'un évènement : `crash(g3,main(1))` empêche `run(train,main(2))` et donc aussi `run(train,main(3))` et `crash(g1,main(3))`.
Inversement, `crash(g1,main(3))` est empêché par `crash(g3,main(1))` et l'est donc aussi par `push(g3,b)`, dont `crash(g3,main(1))` est une conséquence. Il y a multiplicité des causes selon l'étape de la chaine causale jusqu'à laquelle on remonte. On ne cherche pas à isoler une unique cause principale, mais à identifier les évènements qui participent à la réalisation de la conséquence.

```
r(s0,causes,push(g3,b),0,crash(g3,main(1))).                %A
r(s0,prevents,crash(g3,main(1)),2,run(train,main(2))).
r(s0,prevents,crash(g3,main(1)),2,run(train,main(3))).
r(s0,prevents,crash(g3,main(1)),2,crash(g1,main(3))).       %B
r(s0,prevents,push(g3,b),0,crash(g1,main(3))).
...
r(s1,causes,switch(main(0)),0,run(train,side(0))).
r(s1,causes,switch(main(0)),0,crash(g2,side(2))).
r(s1,prevents,switch(main(0)),0,run(train,main(0))).
r(s1,prevents,switch(main(0)),0,crash(g1,main(3))). [...]
```

6. Modèle du Bien

Les modèels des sections précédentes ne contiennent aucune considération éthique de ce qu'il est souhaitable, bon ou juste de faire ou de provoquer. C'est sur les modèles que nous allons maintenant expliciter, le modèle du Bien, puis en section suivante le modèle du Juste, que repose le jugement éthique à proprement parler. En premier lieu, il s'agit de former une évaluation du Bien sur laquelle pourront s'appuyer les théories du Juste pour décider de ce qu'il est admissible ou non de faire dans un contexte donné. Un modèle du Bien doit ainsi fournir une appréciation de la valeur éthique intrinsèque de finalités ou d'évènements. C'est un élément essentiel pour les éthiques conséquentialistes que de définir et quantifier le « Bien ». Le rôle de ce modèle est donc d'informer le modèle du Juste de ce qui est *atomiquement* bon ou mauvais, en précisant éventuellement de quelle façon (à quel degré, vis à vis de quelle cible, selon quelle modalité, etc.). Ainsi, les théories du juste peuvent s'abstraire des détails de cette évaluation souvent dépendante du domaine, et s'intéresser uniquement à la façon de prendre en compte le Bien et le Mal causés directement ou non par l'action jugée.

6.1. Qualifier le Bien

Nous présentons dans cette section deux modes de définition du Bien, relatifs aux droits et aux valeurs. Ces deux approches permettent de raisonner différemment sur les impacts éthiques des évènements dans un domaine donné. Les droits, les valeurs et les autres moyens de définir le Bien sont appelés *modalités*. De plus, nous identifions aussi la cible de cette modalité. Ainsi, nous indiquons qu'un évènement donné est bon ou mauvais pour une cible donnée selon une modalité donnée par les prédicats `good(E,X,M)` et `bad(E,X,M)` où `E` est un évènement, `X` une cible et `M` une modalité. Il est ainsi possible qu'un évènement soit à la fois bon et mauvais, vis-à-vis de cibles ou de modalités différentes. Par exemple, une transplantation de rein peut être considérée comme bonne pour le droit à la santé de la personne qui le reçoit, mais mauvaise pour celui de celle qui le donne. De même, révéler une vérité blessante pourrait être bon vis-à-vis d'une valeur comme l'honnêteté mais mauvais selon des valeurs de prévenance ou de tact. Inversément, selon les modalités considérées, il est possible qu'un évènement ne soit ni bon ni mauvais pour qui que ce soit.

6.1.1. Droits

L'*utilitarisme des droits* de Nozick postule que le Bien à maximiser consiste en la non violation des droits (Nozick, 1974). Un droit peut être défini comme une "*revendication justifiée que les individus et les groupes peuvent faire sur d'autres individus ou sur la société; avoir un droit c'est être en mesure de déterminer par ses choix, ce que les autres doivent ou ne doivent pas faire*" (Beauchamp, Childress, 2001). Cette définition capture le fait qu'un droit indique à la fois un état des choses pour la personne concernée (l'exercice du droit) et une contrainte imposée aux autres (l'interdiction de violer le droit). Nous définissons les règles de sorte qu'un évènement violant le droit d'une ou plusieurs personnes est *mal* par rapport à ce droit. Pour ce qui est de définir le *Bien* vis à vis des droits, nous estimons que ne pas violer un droit n'est pas suffisant. Seront considérés comme bons les actes qui rétablissent une condition correspondant à ce droit (par exemple soigner quelqu'un rétablit son droit à la santé). De plus, le Bien découle aussi de l'empêchement d'évènements mauvais.

Nous proposons ici d'associer chaque droit à un ensemble de fluents correspondant aux états qu'il est censé protéger ou éviter. Ainsi le droit à la sécurité pourrait s'interpréter comme le droit d'être vivant et de ne pas être blessé. Ces droits sont définis par `right(M,X,F)` où `M` est la modalité (le nom du droit), `X` une personne ou un groupe et `F` un fluent relatif à un état de `X`. Ces déclarations se font dans la *spécification de modalités*. Ainsi, dans le cadre de l'exemple du trolley, il suffira de définir les droits avec `right(life,X,alive(X)):-group(X).` dans cette spécification pour représenter un droit à la vie. La théorie du Bien est alors définie par les règles suivantes, considérant les cas d'infraction selon que le droit garantit un état positif ou négatif et le cas de rétablissement de façon directe (la règle restant valide si `F` est de la forme `neg(r)`).

```
bad(E,X,M):-effect(E,neg(F)),right(M,X,F).
```

```
bad(E,X,M):-effect(E,F),right(M,X,neg(F)).
good(E,X,M):-effect(E,F),right(M,X,F).
```

6.1.2. Valeurs

Une théorie fondée sur les valeurs fournit également un moyen efficace d'évaluer le mérite initial des évènements. Une valeur peut être définie comme *"une conception, explicite ou implicite, distinctive d'un individu, ou caractéristique d'un groupe, de ce qui est souhaitable et qui influence la sélection des modes, des moyens et des fins d'action disponibles."* (Kluckhohn, 1951). Une valeur est donc un type d'entité indépendante que peuvent assumer les actions et leurs conséquences. Nous définissons ainsi les règles de sorte qu'un évènement qui démontre l'expression d'une valeur particulière est *Bien* par rapport à cette valeur, et un évènement qui démontre la négation d'une valeur est *Mal* par rapport à cette valeur. Les autres évènements sont considérés ni bons ni mauvais par rapport à celle-ci. Notez que les valeurs sont définies par `value(M)` dans la *spécification de modalités* et qu'il est aussi nécessaire d'y définir en fonction du domaine et de la valeur ce que constitue la démonstration ou non d'une valeur. La théorie du Bien est alors minimaliste, la charge reposant essentiellement sur la spécification de modalités que l'on illustre ensuite avec notre exemple médical.

```
bad(E,X,M):-displays(E,X,neg(M)),value(M).
good(E,X,M):-displays(E,X,M),value(M).
```

Exemple 2(5) : spécification de valeurs dans le dilemme médical

Le docteur considère ici la propension à venir en aide (`helpfullness`) comme une valeur pour définir du Bien, que tout acte qui met fin à la maladie d'un patient démontre cette valeur, et que tuer le patient est à l'opposé de cette valeur. Pour illustrer la possibilité de considérer plusieurs valeurs, nous considérons aussi une valeur de respect des morts (`respectDead`) qui est enfreinte par la transplantation.

```
value(helpfulness).
displays(E,X,neg(helpfulness)):-effect(E,neg(alive(X))).
displays(E,X,helpfulness):-effect(E,neg(sick(X))).
value(respectTDead).
displays(transplant(X1,X2),X1,neg(respectDead)):-event(transplant(X1,X2)).
```

6.2. Quantifier le Bien

Une fois déterminé le contenu du Bien, nous procédons à sa quantification, c'est-à-dire à la « pesée » des bonnes et mauvaises ramifications des évènements. Cette projection dans une échelle numérique est classique des approches utilitaristes et pose de nombreuses questions quand à la détermination des différents poids. Nous définissons deux principaux paramètres:

– l'importance des cibles impliquées. Ce paramètre est relatif au groupe ou à l'individu affecté par l'évènement. Cette information est donnée par le X dans les prédicats `good(E,X,M)` et `bad(E,X,M)`. Dans le cas de groupes d'individus, il est

courant de considérer le nombre de personnes impliquées dans l'évènement. Ce paramètre est pris en compte par l'attribution à chaque groupe d'une valeur numérique, exprimée dans le prédicat `t_Weight(E,G,N)` où `E` est un évènement, `G` son groupe cible et `N` le poids donné.

– l'importance de la modalité affectée par l'évènement. Par exemple, être bienfaisant est peut être plus important qu'être poli, respecter le droit à la vie est peut-être plus important que respecter le droit de propriété. De plus, il peut être nécessaire de différencier les poids associés au Bien et au Mal selon cette modalité, car il est souvent considéré plus grave de faire le Mal que de ne pas faire le Bien. Ce paramètre est pris en compte par l'attribution à chaque modalité d'une valeur numérique, exprimé dans le prédicat `m_Weight(M,Ng,Nb)` où `M` est la modalité et `Ng`, `Nb` sont les poids donnés respectivement au fait de faire le Bien ou le Mal selon cette modalité.

L'attribution de poids aux modalités et aux groupes est non triviale et la méthode proposée ici fait acte d'introduction. Il est possible, par exemple, de l'enrichir en prenant en compte d'autres dépendances, telles que la corrélation entre certaines modalités et personnes (l'autonomie pourrait être essentielle pour les adultes et la sécurité pour les enfants). Nous proposons ici simplement de combiner ces poids de façon multiplicative. Il est à noter que les poids de cible et de modalité sont définis dans la *spécification de modalités*. Ces poids constituent ainsi une première étape, et pourront être agrégés de différentes façon par les théories conséquentialistes.

```
wGood(E,N1*N2,X,M):-good(E,X,M), t_Weight(E,X,N1),m_Weight(M,N2,_).
wBad(E,N1*N2,X,M):- bad(E,X,M), t_Weight(E,X,N1),m_Weight(M,_,N2).
```

Dans nos deux exemples, les poids des modalités seront définis par de simple faits (par exemple `m_Weight(life,1,1)` pour le trolley), et nous utiliserons la règle suivante pour déterminer le poids des groupes par le nombre de personnes qu'ils contiennent: `t_weight(X,N):-numberInGroup(X,N).`

7. Modèle du Juste

Le modèle du Juste a la tâche de déterminer les actions justes, c'est à dire admissibles, selon les circonstances de leurs exécutions. Il s'appuie sur les informations fournies par le reste du système et applique une ou plusieurs théories du Juste pour juger des actions. Ce modèle permet ainsi d'obtenir les actions admissibles selon les différents principes éthiques en inférant le prédicat `per(J,A)`, qui signifie que l'action `A` est admissible (permissible) selon la théorie `J` (les différents principes éthiques étant identifiés par le prédicat `th(J)`). En pratique, la plupart des théories du Juste se concentrent sur la définition des actions non admissibles, nous définissons l'admissibilité à partir de la non-admissibilité (indiquée par le prédicat `imp(J,A)`) de façon directe par la règle : `per(J,A):-act(A),not imp(J,A),th(J).`

7.1. Éthiques conséquentialistes

L'éthique conséquentialiste existe sous de nombreuses formes, allant des principes simples d'action aux théories complexes pour maximiser le Bien. Nous en décrivons et modélisons cinq, après une discussion sur l'agrégation des poids.

7.1.1. Agrégation des poids

Les approches conséquentialistes évaluent une action par agrégation de biens ou de maux atomiques, s'appuyant sur les valeurs cumulées de ses conséquences. Les différentes façons d'agréger ces effets sont à la base de différentes variantes d'utilitarisme. Nous prenons ici une approche classique rassemblant par somme l'ensemble des conséquences d'une action, amalgamant ainsi en un seul score les différentes modalités et personnes affectées. Une approche avec un souci d'équitabilité pourrait considérer de sommer ces poids par individu avant de les agréger avec un minimum, mesurant le Bien par le Bien de l'individu le moins bien loti. Si l'on utilise des relations causales de prévention, considèrant qu'un Bien évité participe du Mal, on somme donc le Bien causé et le Mal évité pour déterminer le Bien total et inversément pour le Mal. Si on utilise uniquement $\mathbb{C}_b$, il suffit d'ignorer cet aspect (`N2` vaut 0):

```
wBadAct(A,N1+N2):-act(A),number(N1;N2),
        N1=#sum{K,X,M,E:wBad(E,K,X,M),r(S,causes,A,T,E)},
        N2=#sum{K,X,M,E:wGood(E,K,X,M),r(S,prevents,A,T,E)}.
wGoodAct(A,N1+N2):-act(A),number(N1;N2),
        N1=#sum{K,X,M,E:wGood(E,K,X,M),r(S,causes,A,T,E)},
        N2=#sum{K,X,M,E:wBad(E,K,X,M),r(S,prevents,A,T,E)}.
weightAct(A,Ng-Nb):-act(A),number(Ng;Nb),wBadAct(A,Nb),wGoodAct(A,Ng).
```

7.1.2. Proscription d'actions purement préjudiciables

Le premier principe conséquentialiste affirme que les actions ayant des effets purement préjudiciables sont inadmissibles. Cette règle intuitive est pertinente pour la plupart des scénarios éthiques et peut compléter d'autres théories du Juste qui ne se concentrent que sur des actions à effets complexes. Pour mettre en œuvre cette règle, nous définissons les prédicats `badCons(S,A,T)` et `goodCons(S,A,T)`, qui indiquent respectivement qu'une action `A` qui se produit à `T` dans `S` provoque au moins une mauvaise ou une bonne conséquence (ou inversément évite au moins une bonne ou une mauvaise conséquence). Nous déclarons ensuite qu'une action est inadmissible si elle n'a que des mauvaises conséquences.

```
badCons(S,A,T):-act(A),r(S,causes,A,T,E),bad(E,X,M).
goodCons(S,A,T):-act(A),r(S,causes,A,T,E),good(E,X,M).
%-- gestion prevents (ignoré si moteur causal basique)
badCons(S,A,T):-act(A),r(S,prevents,A,T,E),good(E,X,M).
goodCons(S,A,T):-act(A),r(S,prevents,A,T,E),bad(E,X,M).
th(pBad).
imp(pBad,A):-badCons(S,A,T),not goodCons(S,A,T).
```

7.1.3. Principe de la moins mauvaise conséquence

Également appelé *maximum minimorum*, ce principe déclare qu'une action est inadmissible si sa pire conséquence est pire que la pire conséquence de toute autre action possible. Pour formaliser cette règle, nous déterminons d'abord une hiérarchie entre les conséquences des actions, afin d'ensuite indiquer la pire. Le prédicat `worse(E1,E2)` indique que la conséquence E1 d'une action est pire que la conséquence E2 de la même ou d'une autre action. Le prédicat `worst(S,A,T,E)` détermine alors la limite basse d'un ordre induit par le prédicat `worse`. Enfin, nous déclarons qu'une action A1 est inadmissible si sa pire conséquence E1 est pire que la pire conséquence E2 de toute autre action A2. Toutes les autres actions sont admissibles. Nous donnons ici la définition uniquement pour le moteur causal basique, sans tenir compte de la prévention d'effets bénéfiques.

```
worse(E1,E2):-wBad(E1,N1,_,_),wBad(E2,N2,_,_),N2<N1.
worst(S,A,T,E):-act(A),event(E),r(S,causes,A,T,E),
    {event(E2):r(S,causes,A,T,E2),worse(E2,E),not worse(E,E2)}0.
th(lBad).
imp(lBad,A1):-worse(E1,E2),worst(S1,A1,T1,E1),worst(S2,A2,T2,E2),A1!=A2.
```

La définition de `worse(E1,E2)` prête à discussion. S'appuyer sur chaque conséquence atomique de l'action signifie que l'on s'inquiète de l'importance des mauvaises conséquences indépendemment de leur nombre. Dans notre exemple médical où une même action peut tuer plus ou moins de groupes de tailles identiques, toutes les conséquences seront considérées équivalentes, car on considère la plus mauvaise chose qu'il arrive à un groupe comme conséquence d'un évènement donné (ici, mourir). Nous retrouvons là l'idée de maximiser le bien être du plus mal loti. Dans le trolley, en revanche, les groupes étant de tailles différentes (le crash a pour effet de tuer un groupe de 5 personne plutôt que d'avoir 5 fois l'effet de tuer un individu), ces effets n'ont pas le même poids et c'est l'action qui tue le plus petit groupe qui sera retenue. Le choix de décomposer ou non chaque effet dans le modèle impacte ainsi l'évaluation de ces effets atomiques. Pour compenser ces différences de modèles, on peut définir `worse` en agrégeant tout ou partie des conséquences (par exemple agrégation de tous les maux frappant chaque individu). Pour comparaison, on peut définit le pire par « `worse2(E1,E2):- wBadAct(E1,N1),wBadAct(E2,N2),N2<N1.` ». Les définitions de la théorie résultante (`lBadC`) se simplifient alors en :

```
th(lBadC).
imp(lBadC,A):-wBadAct(A,N),wBadAct(A2,N2),N2<N.
```

7.1.4. Principe d'analyse coût-avantage

Ce principle indique qu'une action est admissible si elle est globalement bénéfique, c'est-à-dire si ses bonnes conséquences l'emportent sur les mauvaises. Nous utilisons le prédicat `weightAct(A,N2)` défini en 7.1.1 pour indiquer que A est inadmissible si ce poids total N est négatif.

```
th(benC).
imp(benC,A):-weightAct(A,N),number(N),N<0.
```

7.1.5. Utilitarisme de l'acte

L'utilitarisme de l'acte exige que l'on évalue une action directement selon le *principe d'utilité*, qui stipule que l'action moralement correcte est celle qui a les meilleures conséquences globales pour le bien-être ou l'utilité de la majorité des parties concernées (Bentham, 2001). Une action est donc admissible si, compte tenu de toutes les autres actions disponibles, elle a les meilleures conséquences dans l'ensemble. En utilisant le prédicat `weightAct` défini en 7.1.1, nous déterminons un ordre de préférence entre les actions dans le domaine et déclarons qu'une action A1 est inadmissible s'il existe une autre action A2 dont le poids est supérieur.

```
th(actU).
imp(actU,A1):-weightAct(A1,N1),weightAct(A2,N2),N1<N2.
```

7.1.6. Utilitarisme de la règle

L'utilitarisme de la règle évalue une action en deux temps (Struhl, Rothenberg, 1975). La première étape consiste à évaluer les règles morales sur la base du principe d'utilité: il s'agit de déterminer si une règle ou un ensemble de règles morales engendrera les meilleures conséquences, supposant que tout ou la plupart des agents s'y plient. Dans la vie quotidienne, de telles règles peuvent inclure « Ne pas voler », ou « Garder ses promesses ». La deuxième étape consiste à évaluer les actions individuelles relativement à ce qui a été justifié au cours de la première étape. Une action n'est admissible que si la règle à laquelle elle appartient respecte le principe d'utilité, outre son propre respect du principe. Par exemple, si « Ne pas voler » est une règle adoptée, le vol sera toujours inadmissible, même dans l'instance ou un vol produirait la plus grande utilité (par exemple, en alimentant un affamé). Le prédicat `ruleCount(R,N)` regroupe tous les poids `N` des actions qui appartiennent à une règle `R`; le prédicat `weightRule(R,N)` les concatène. Nous déclarons alors qu'une action `A` est inadmissible si elle est une instance d'une règle `R` globalement nuisible, c'est-à-dire dont les mauvaises conséquences l'emportent sur les bonnes, considérant toutes ses instanciation. Toute autre action est admissible. Il est à noter que les instances de règles et les règles sont définies par `rule(R)` et `instance(A,R)` dans la *spécification déontologique*.

```
weightRule(R,N) :-rule(R),number(N),
     N=#sum{N1,A:weightAct(A,N1),instance(A,R)}.
th(ruleU).
imp(ruleU,A):-act(A),instance(A,R),weightRule(R,N),N<0.
```

Nous illustrons cela par une spécification déontologique du dilemme médical.

Exemple 2 (6) : règles et instances pour le dilemme médical

Le médecin envisage deux règles morales : « Ne pas soutenir l'expérimentation animale » (`ani`) et « Ne pas utiliser de traitements expérimentaux » (`exp`). La première règle lui interdit de prescrire des mécidaments testés sur des animaux, tandis que la seconde n'autorise l'utilisation d'un traitement que s'il a été testé cliniquement.Ces règles peuvent donc se modéliser comme suit : `give(Z)` est une instance de la règle `ani` si `Z` n'a pas été l'objet de tests sur des animaux et c'est une instance de la règle `exp` si `Z` a été testé cliniquement. Il nous faut de plus rajouter quelques connaissances sur les tests des médicaments considérés. Ici, Alpha n'a pas été testé sur des animaux alors que Beta et Gamma si, et tous les produits testés sur des animaux ont été aussi été testés cliniquement. L'administration du traitement Beta ou Gamma est donc une instance de `exp` tandis que celle de Alpha est une instance de `ani`.

```
rule(ani;exp).
test((beta;gamma),animal).
test(Z,clinic):-test(Z,animal),treatment(Z).
instance(give(Z),ani):-treatment(Z),not test(Z,animal).
instance(give(Z),exp):-treatment(Z),test(Z,clinic).
```

7.2. Éthiques déontologiques

Dans cette section, nous présentons trois doctrines déontologiques. Deux d'entre elles sont purement déontologiques, les codes de conduite et l'éthique kantienne. La doctrine du double effet comporte des contraintes conséquentialistes.

7.2.1. Codes de conduite

Un code de conduite est un ensemble de règles qui décrit les obligations, les interdictions ou les responsabilités d'un individu, d'un groupe ou d'une organisation. Il spécifie les principes qui guident la prise de décision ou les procédures de ceux qui sont contraints par le code. Les codes de conduite varient dans leur portée et leur nature, allant des codes déontologiques professionnels aux commandements religieux. Le comportement et la moralité sont typiquement déterminés par un corps global, tel qu'une entreprise, un état ou un dieu. Nous illustrons ici ce type de contrainte en modélisant une règle commune qu'est l'interdiction de tuer. Nous l'illustrons de deux façons selon qu'on l'on se focalise un évènement donné (`kills` pour `dNK`) ou sur les effets (`neg(alive)` pour `dNK2`).

```
th(dNK;dNK2).
imp(dNK,A):-act(A),r(S,causes,A,T,kills(_,_,_)).
imp(dNK2,A):-act(A),r(S,causes,A,T,E),effect(E,neg(alive(G))).
```

Comme on le voit, il peut s'agir de règles spécifiques au domaine (présence du fluent `alive/1` ou de l'évènement `kills/3`). En pratique on choisira plutôt de placer dans la spécification déontologique une série de règles réprésentant les interdictions instaurées par le code de conduite. Cela suppose cependant d'expliciter les

règles et de les modéliser manuellement, ce qui peut être coûteux. Les nombreux travaux sur les normes et la manière de les représenter, extraire, éliciter ou apprendre sont alors pertinents pour des approches déontologiques de l'éthique. L'utilisation de normes est en effet un moyen de rendre compte de règles de conduite qui peut être utilisé pour traduire des codes moraux (Serramia *et al*., 2018). Les règles pouvant être contradictoires, des mécanismes de résolution de conflit peuvent complèter de tels codes. La représentation d'obligations et d'interdiction, concepts clefs des approches déontologiques, est également centrale dans les logiques déontiques. Les travaux de Bringsjord sur le calcul des évènements déontique sont un bon exemple de leur utilisation dans un cadre de modélisation éthique (Govindarajulu, Bringsjord, 2017).

7.2.2. La formule de la fin en elle-même

Cette formule est un élément d'éthique kantienne qui met l'accent sur la valeur intrinsèque de la vie humaine. C'est un impératif moral qui interdit d'utiliser les personnes comme moyen pour d'autres fins, les personnes étant des fins en elles-mêmes en vertu de leur nature d'êtres rationnels (Kant, 1964). La formule contraste la valeur intrinsèque, qui est persistante et souveraine, avec la valeur instrumentale, qui dépend de ce qu'elle produit. Nous présentons le prédicat `aim(A,E)` qui indique que le but de l'action A est de provoquer l'évènement E et utilisons le fluent `involves(E,X)` pour indiquer qu'au moins une personne est impliquée dans E. En première approximation, nous définissons `involves(E,X)` à partir des effets négatifs. Cependant cette notion n'y est pas réductible et peut faire l'objet de spécifications plus précises dans un domaine donné. Nous déclarons alors qu'une action A est inadmissible si elle provoque un évènement E qui implique au moins une personne, mais où E n'est pas un but de A. Toute autre action est admissible. Les buts sont définis par `aim(A,E)` dans la *spécification déontologique*. Dans le cas du dilemme médical, nous utilisons « `aim(give(Z),E):-effect(E,neg(sick(X))),event(give(Z)).` ».

```
involves(E,X):-bad(E,X,_).
th(kant).
imp(kant,A):-act(A),r(S,causes,A,T,E),involves(E,X),not aim(A,E).
```

7.2.3. La doctrine du double effet

La doctrine du double effet est un ensemble de critères éthiques utilisés pour évaluer la permissibilité éthique d'une action qui a à la fois de bonnes et de mauvaises conséquences. Elle dicte qu'une personne peut licitement exécuter une action en sachant qu'elle aura des bons et mauvais effets, à condition que : 1) L'action elle-même soit bonne ou moralement neutre ; 2) Le mauvais effet ne soit pas directement voulu ; 3) Le bon effet résulte de l'acte et non du mauvais effet ; 4) Le bon effet soit plus important ou égal au mauvais effet (Foot, 1967). la première règle proscrit une action si elle est intrinsèquement mauvaise, correspondant à la condition 1. Les deux suivantes proscrivent une action si elle provoque un mauvais effet qui conduit à un bon effet (ou en évite un mauvais). Cela correspond aux conditions 2 et 3, en l'absence d'intentions explicites. Enfin la dernière règle traduit la condition 4 en rejetant une action si son effet global est mauvais, ce qui équivaut au principe d'analyse coût-avantage.

```
th(dde).
imp(dde,A):-act(A),bad(A,X,M).
imp(dde,A):-act(A),r(S,causes,A,T,E1),r(S,causes,E1,T2,E2),
     bad(E1,X1,M1),good(E2,X2,M2).
imp(dde,A):-act(A),r(S,causes,A,T,E1),r(S,prevents,E1,T2,E2),
     bad(E1,X1,M1),bad(E2,X2,M2).
imp(dde,A):-imp(benefitsCosts,A).
```

8. Comparaison des principes éthiques

Les différents principes éthiques présentés dans la section précédente s'appuient sur des critères différents pour déterminer les actions admissibles et peuvent ainsi aboutir à des conclusions très différentes. Nous illustrons cela sur nos deux exemples avant de discuter des grandes similitudes ou divergences des principes présentés ici.

8.1. Comparaison sur les exemples illustratifs

Nous illustrons la manière dont chaque contrainte éthique décrite ci-dessus gère un dilemme éthique à l'aide des deux exemples utilisés tout au long de l'article. Ces derniers sont éclairants car il sont conçus pour poser des choix difficiles différentiant les différentes approches. Typiquement, le dilemme du Trolley a été construit pour remettre en question les approches purement conséquentialistes en introduisant des différences sur les moyens non prises en compte par celles-ci.

Dilemme Médical

Ici, nous utilisons le moteur causal basique et un modèle du Bien basé sur la valeur `helpfulness` pondérée à 1 (`m_weight(helpfulness,1,1)`). Un code de conduite supplémentaire interdisant les transplantations est aussi donné :

```
th(deon).
imp(deon,A):- act(A),r(S,causes,A,T,transplant(X1,X2)).
```

Le gain en termes de vies sauvées (c'est-à-dire les patients guéris moins les patients tués) pour chaque traitement est donc : **Alpha -5**; **Beta 5**; **Gamma 20**. Les résultats de l'évaluation éthique sont résumés ci-dessous.

	pBad	**lBad**	**lBadC**	**benC**	**actU**	**ruleU**	**dNK**	**deon**	**kant**	**dde**
Alpha	Oui	Oui	Oui	Non	Non	Non	Non	Oui	Non	Non
Beta	Oui	Oui	Non	Oui	Non	Oui	Non	Oui	Non	Oui
Gamma	Oui	Oui	Non	Oui	Oui	Oui	Non	Non	Non	Non

On observe tout d'abord que quasiment toutes les combinaisons d'admissibilité ou non des trois actions possibles sont couvertes par au moins un des principes éthiques présentés : seul le cas où Alpha et Gamma serait admissible mais pas Beta ne correspond à aucun des principes présentés. Quand Beta est rejeté, il y a toujours au moins

un autre traitement qui est aussi rejeté. Cette variété des conclusions illustre que sur un exemple complexe comme celui-ci, le choix du principe éthique suivi est décisif. Alpha sera choisi si l'objectif est de minimiser le nombre total de morts (`lBadC`), Gamma le sera s'il s'agit d'avoir le meilleur gain en terme de vies sauvées (`actU`) et enfin Beta sera la seule solution admise par la doctrine du double effet (`dde`) qui exige certaines conditions sur les moyens tout en demandant un gain positif.

Comparons d'abord Alpha et Beta, qui sont de structure similaire (ils tuent certaines variantes et en sauvent d'autres directement), mais diffèrent par le fait que Alpha fait moins de victimes en tout (20 morts contre 25 morts pour Beta) alors que que Beta a un meilleur gain qu'Alpha en terme de vie sauvées (5 pour Beta contre -5 pour Alpha) : c'est-à-dire que Beta fait plus de victimes mais compense mieux ces pertes. Alpha l'emporte sur Beta (i.e. est admissible sans que Beta ne le soit) uniquement pour le principe `lBadC`, qui cherche à minimiser le mal produit sans considérer le bien produit en compensation. A l'inverse, Beta l'emporte sur Alpha pour les principes `benC` et `dde` qui prennent en compte ces compensations ainsi qu'avec `ruleU` où Alpha et Beta sont considérés comme instances de règles différentes. Si l'on compare Beta et Gamma, on retrouve une opposition entre conséquentialisme pur et déontologie : Gamma l'emporte selon l'utilitarisme de l'Acte (`actU`), qui maximise le Bien total sans considération des moyens, là où Beta l'emporte sur Gamma quand le moyen de la transplantation est remis en cause : soit directement (`deon`), soit comme utilisation de la mort d'autrui pour sauver une vie (`dde`).

Notons enfin que, dans notre exemple illustratif, quelques principes ne sont pas différenciés. Ainsi `pBad` et `lBad` admettent tous deux tous les traitements. Dans le premier cas, c'est dû au fait que chacun des traitements sauve au moins quelques personnes et a donc des effets positifs. Si l'on rajoutait un traitement Delta qui provoque la mort d'une variante sans soigner aucune variante, il serait rejeté par `pBad`, mais pas par `lBad`. En effet ce dernier regardant les effets négatifs de chaque évènement atomique (mort d'un groupe) sans les agréger, il conclut que la pire conséquence est toujours qu'un groupe meurt. Mais comme elle est atteinte dans tout les cas ces actions, non différentiées, sont toutes également admissibles. Par contre, si on ajoutait un traitement Eta qui soigne une seule variante, mais sans tuer qui que ce soit, `pBad` continuerait s'admettre tous les traitements (sauf Delta) alors que `lBad` n'admettrait plus que Eta, rejettant tous les autres traitements. `pBad` et `lBad` sont donc bien des principes différents. De même `dNK` et `kant` rejettent ici tous deux toutes les actions, mais il est facile d'imaginer un contexte moins meurtrier (par exemple des cas de vol ou de mensonge) où le principe de la fin en elle-même rejette des cas qui ne seraient pas concernés par l'injonction de ne pas tuer.

Pour montrer l'influence du modèle du Bien, nous prenons maintenant aussi en compte la valeur `respectDead` avec la même pondération, pénalisant donc le traitement Gamma. Les résultats affectés sont indiqués ci-dessous en gras.

	pBad	lBad	lBadC	benC	actU	ruleU	dNK	deon	kant	dde
Alpha	Oui	Oui	Oui	Non	Non	Non	Non	Oui	Non	Non
Beta	Oui	Oui	Non	Oui	**Oui**	**Non**	Non	Oui	Non	Oui
Gamma	Oui	Oui	Non	**Non**	**Non**	**Non**	Non	Non	Non	Non

Bien que cette valeur n'affecte strictement que le traitement Gamma, cela a un effet sur Beta pour les principes `actU` et `ruleU`, qui devient admissible par élimination des meilleures options. Remarquons que `benC` et `ruleU`, qui avaient les mêmes conclusions précédemment, se différencient ici.

Problème du Trolley

Pour ce dilemme, nous utilisons le moteur causal avec prévention et un modèle du Bien basé sur le droit à la vie. Nous nous en tenons aux théories gérant explicitement la prévention. Pour rappel, ici, push tue une personne alors que switch en tue deux. Les résultats sont bien cohérents avec les conclusions pour lesquelles cette expérience de pensée a été construite (i.e. la comparaison entre `actU` et `dde`). La doctrine du double effet rejette le push là où les doctrines purement utilitaires le préfèrent. La doctrine du double effet est ici la seule des doctrine que nous avons modélisé à parvenir à la conclusion considérée comme intuitive par ses défenseurs (Foot, 1967).

	pBad	lBadC	benC	actU	dNK2	dde
Push	Oui	Oui	Oui	Oui	Non	Non
Switch	Oui	Non	Oui	Non	Non	Oui

Temps d'éxécution

Nous donnons ci-dessous les temps d'exécution obtenus pour nos deux exemples sur un poste équipé d'un processeur multicoeur à 3.60GHz $\times 8$ avec 8 Go de RAM. On compare une éxécution non modulaire, prenant l'union de tous les modèles pour en dériver en un seul appel au solveur toutes les conclusions avec l'approche modulaire, qui procède par étape en dérivant d'abord la trace d'évènements, puis la trace causale et enfin les admissibilités. L'approche modulaire est 8 à 14 fois plus rapide ici. Dans le cas du dilemme médical, en pratique, elle consacre 1.6s à obtenir la trace d'évènements, 0.04s à dériver la trace causale et 0.16s à conclure sur les toutes admissibilités. L'essentiel du temps est donc consacrée à obtenir la trace évènementielle, étape nécessaire à tous les principes éthiques. Les éventuelles différences de coûts entre les calculs des principes présentés apparaissent ainsi négligeables par rapport à cela.

Exemple	Exec. non modulaire	Exec. modulaire
Dilemme médical	14 s	1.8s
Problème du Trolley	1.7s	0.12s

8.2. Discussion

Si tous les principes éthiques que nous avons présentés ici sont bien différents, la modélisation de ces théories souligne certaines caractéristiques que nous présentons ici avant d'introduire une notion de comparaison entre deux principes.

8.2.1. Caractérisations des théories

Un premier critère primordial est de savoir si l'existence d'autres options impacte ou non le jugement porté sur une action. Les théories du Juste qui considèrent que cela doit être le cas doivent évaluer chaque action par rapport aux autres actions possibles. Il s'agit de principes *relatifs*. Cela se manifeste dans les règles qui les expriment par le fait que l'on fait intervenir d'autres actions (issues de simulations différentes) dans le jugement d'une action donnée. C'est ce type de théories qui force l'utilisation explicite de simulations dans notre cadre. Les principes `lBad`, `lBadC`, `actU` et `ruleU` sont de ce genre. Cela se constate directement pour les trois premiers dont les règles d'inadmissibilité font intervenir une autre action (variable `A2`). L'*utilitarisme de la règle* (`ruleU`) est particulier en ce sens que la permissibilité de *toutes* les actions est déterminée selon l'impact de chacune individuellement. Les autres actions interviennent en pratique à travers la somme qui est faite des poids de toutes les instances d'une règle. L'ajout d'un nouveau choix possible peut conduire à reconsidérer l'admissibilité des options précédemment testées.Les conclusions des principes relatifs sont donc *révisables* (par rapport à l'ajout d'alternatives). À l'inverse, les principes *indépendants* évaluent chaque action sans la comparer aux options disponibles. Ainsi, leur conclusions restent inchangées par l'ajout d'autres alternatives. Toutes les théories du Juste déontologiques présentées ici sont de ce genre, ainsi que `pBad` et `benC`.

Une autre caractérisation est de savoir si le principe s'appuie sur un critère de *maximisation* ou sur un critère de *satisfaction* pour juger de l'admissibilité d'une action. Un critère de satisfaction établit des conditions invariables d'admissibilité que doit satisfaire une alternative tandis qu'un jugement basé sur un critère de maximisation revient à comparer toutes les alternatives selon un ordre donné (par exemple l'ordre induit par l'aggrégation des quantifications operées par la théorie du Bien, dans le cas de l'utilitarisme de l'acte) et à ne considérer comme admissibles que les alternatives maximales pour cet ordre (il peut typiquement y en avoir plusieurs en cas d'ex aequo ou d'ordre partiel). Comme cela suppose une comparaison des alternatives, un principe basé sur un critère de maximisation est forcément relatif. La réciproque n'est cependant pas vraie. Par exemple, bien qu'il s'agisse d'un principe relatif, l'utilitarisme de la règle repose sur un critère de satisfaction. En effet, il exige que la règle dont l'action considérée est une instance se soit pas globalement nuisible, ce qui se traduit par un critère absolu (rejet si `N<0`) sur la quantification des effets de toutes les instances de la règle (`weighRule(R,N)`). Une propriété intéressante des principes basés sur un critère de maximisation est qu'il sont *non-bloquants*, c'est-à-dire qu'ils considèrent toujours au moins une action comme admissible parmi l'ensemble des actions possibles. A l'inverse, lorsqu'on se base sur un critère de satisfaction, il est possible de rejeter toutes les options (voir `kant` et `dNK` en 8.1).

Un dernier critère important, qui différentie généralement les approches conséquentialistes et déontologiques est de savoir si les étapes et évènements intermédiaires influent sur le jugement ou si seule la situation finale est considérée. Si seule l'issue importe, le principe est considéré comme *purement conséquentialistes* et des raisonnements du type « la fin justifie les moyens » sont valides. C'est évidemment le cas

de l'utilitarisme de l'acte, de celui de la règle et du principe coût-avantage et comme on pouvait l'attendre, ce n'est pas celui des principes déontologiques. Par contre, du fait qu'elles se focalisent sur l'observation du pire en ignorant le Bien qui pourrait le compenser, les deux interprétations proposées du *principe de la moins mauvaise conséquence* ne sont pas uniquement sensibles à l'état final. Le mal produit dans l'atteinte de ce résultat est considéré sans phénomène de compensation.

Le tableau suivant récapitule cela en indiquant pour chaque principe présenté s'il est relatif (ou indépendant), basé sur un critère de satisfaction (Sat.) ou de maximisation (Max.) et purement conséquentialiste ou non (P. cons).

	pBad	**lBad**	**lBadC**	**benC**	**actU**	**ruleU**	**dNK**	**deon**	**kant**	**dde**
Relatif ?	Non	Oui	Oui	Non	Oui	Oui	Non	Non	Non	Non
Critère	Sat.	Max.	Max.	Sat.	Max.	Sat.	Sat.	Sat.	Sat.	Sat.
P. cons. ?	Oui	Non	Non	Oui	Oui	Oui	Non	Non	Non	Non

8.2.2. Comparaisons de principes éthiques

Il est important de noter que différentes théories du Juste peuvent, et parfois doivent, se compléter. Par exemple, dans son acception la plus stricte, la *doctrine du double effet* ne dit rien sur les actions dont les effets sont purement bon ou mauvais et peut être utilement complétée par un principe conséquentialiste pour y remédier. C'est ce que nous avons fait en l'appuyant sur le principe coût-avantage, qui en plus de traduire la contrainte de proportionalité, permet de gérer les cas plus simple où l'on ne retrouve pas deux effets contradictoires. Tous les principes ne sont en effet pas entièrement indépendants et certains peuvent être reliés par une relation d'ordre partiel selon la force de leur conclusions.

DÉFINITION 5. — *Soient une théorie causale* $\mathbb{C}$ *et deux théories du Juste* $\mathbb{R}_1$ *et* $\mathbb{R}_2$ *définissant respectivement des principes éthiques* j_1 *et* j_2.

j_1 *est* plus exigeant *que* j_2 *(noté* $j_1 \sqsubseteq j_2$*) ssi pour tout contexte* Ctx*, théorie du bien* $\mathbb{G}$ *associée (compatible avec* $\mathbb{R}_1$ *et* $\mathbb{R}_2$*) et ensemble d'actions* $\mathcal{A}_c$ *on a :* $Admissible(\mathbb{F}_1, j_1, \mathcal{A}_c) \subseteq Admissible(\mathbb{F}_2, j_2, \mathcal{A}_c)$*, où* $\mathbb{F}_i = \langle \mathbb{A}_{Ctx}, \mathbb{C}, \mathbb{G}, \mathbb{R}_i \rangle$.

Cela signifie qu'un principe j_1 est plus exigeant qu'un autre principe j_2, si, dans une situation équivalente (même contexte, théorie du bien et actions considérées), toute action admissible pour j_1 l'est forcément aussi pour j_2. Dans les théories modélisées dans cet article, la Doctrine du Double Effet est plus exigeante que le principe coût-avantage, qui est lui même plus exigeant que la proscription d'actions purement préjudiciables. On a `dde` $\sqsubseteq$ `benC` et `benC` $\sqsubseteq$ `pBad`.

9. Conclusion

Le cadre présenté ici adapte et s'appuie sur calcul des évènements pour permettre la modélisation de théories éthiques et de scénarios dans lesquels les appliquer. Défini en programmation logique, il présente une méthode et une implémentation de cette

méthode. L'accent est mis sur la hiérarchie et la représentation explicite des processus de raisonnement qui déterminent la prise de décision éthique. Celles-ci permettent d'engendrer des règles avec un fort potentiel expressif qui confèrent aux agents la capacité de décider et d'expliquer leurs décisions, mais aussi de raisonner sur les actions d'autres agents. En outre, la confrontation de théories éthiques avec les contraintes logiques des langages de programmation éclaire ces théories, clarifiant leurs concepts, les relations qui les lient et les ambiguïtés potentielles qu'elles peuvent contenir. Nous envisageons un certain nombre de directions futures pour développer le cadre actuel. Tout d'abord, nous cherchons à modéliser l'intention, qui n'est pour l'instant traitée qu'implicitement, et de modéliser les désirs des agents. Cela permettra aux agents de gérer des scénarios plus complexes et plus réalistes. En outre, nous avons l'intention de permettre la formulation de plans éthiques d'actions dans lesquels plus d'une action peut être évaluée dans une simulation, en travaillant vers un véritable domaine de planification. Enfin, nous visons à intégrer le cadre dans une architecture d'agent, pour pouvoir aussi étudier la confrontation de ces préincipes dans un système multi-agents.

Remerciements

Les auteurs remercient l'Agence Nationale de la Recherche (ANR) pour sa contribution financière sous la référence ANR-13-CORD-0006.

Bibliographie

Alexander L., Moore M. (2016). Deontological ethics. In E. N. Zalta (Ed.), *The Stanford encyclopedia of philosophy.*

Anderson M., Anderson S. (2011). *Machine ethics*. Cambridge University Press.

Anderson M., Anderson S. L. (2014). GenEth: A general ethical dilemma analyzer. In *AAAI*, p. 253–261. AAAI Press.

Arkin R. (2009). *Governing lethal behavior in autonomous robots*. CRC Press.

Beauchamp T., Childress J. (2001). *Principles of Biomedical Ethics*. Oxford University Press.

Beebee H., Hitchcock C., Menzies P. (2009). *The Oxford handbook of causation*. Oxford University Press.

Bentham J. (2001). *A fragment on government*. The Lawbook Exchange, Ltd.

Berreby F., Bourgne G., Ganascia J.-G. (2015). Modelling moral reasoning and ethical responsibility with logic programming. In *LPAR*, p. 532–548.

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

Blass J. A., Forbus K. D. (2015). Moral decision-making by analogy: Generalizations versus exemplars. In B. Bonet, S. Koenig (Eds.), *AAAI*, p. 501–507. AAAI Press.

Bringsjord S., Taylor J. (2012). The divine-command approach to robot ethics. *Robot Ethics: The Ethical and Social Implications of Robotics', MIT Press, Cambridge, MA*, p. 85–108.

Bryson J. J. (2018). Patiency is not a virtue: the design of intelligent systems and systems of ethics. *Ethics Inf. Technol.*, vol. 20, n° 1, p. 15–26.

Cointe N., Bonnet G., Boissier O. (2016). Ethical judgment of agents' behaviors in multi-agent systems. In *AAMAS*, p. 1106–1114. IFAAMAS.

Dennis L., Fisher M., Slavkovik M., Webster M. (2016). Formal verification of ethical choices in autonomous systems. *Robotics and Autonomous Systems*, vol. 77, p. 1–14.

Dignum V. (2017). Responsible autonomy. In *Proceedings of IJCAI*, p. 4698–4704. ijcai.org.

Dodig Crnkovic G., Çürüklü B. (2012, 01 Mar). Robots: ethical by design. *Ethics and Information Technology*, vol. 14, n° 1, p. 61–71.

Feinberg J. (1970). Doing & deserving; essays in the theory of responsibility.

Foot P. (1967). The problem of abortion and the doctrine of double effect. *Oxford Review*, vol. 5, p. 5–15.

Fox M., Long D. (2003, décembre). PDDL2.1: An extension to PDDL for expressing temporal planning domains. *J. Artif. Int. Res.*, vol. 20, n° 1, p. 61–124.

Ganascia J.-G. (2007). Modelling ethical rules of lying with Answer Set Programming. *Ethics and information technology*, vol. 9, n° 1, p. 39–47.

Ganascia J.-G. (2015). Non-monotonic resolution of conflicts for ethical reasoning. In *A construction manual for robots' ethical systems*, p. 101–118. Springer.

Gelfond M. (2008). Answer sets. *Foundations of Artificial Intelligence*, vol. 3, p. 285–316.

Gelfond M., Lifschitz V. (1988). The stable model semantics for logic programming. In *ICLP/SLP*, vol. 88, p. 1070–1080.

Gelfond M., Lifschitz V. (1991). Classical negation in logic programs and disjunctive databases. *New generation computing*, vol. 9, n° 3-4, p. 365–385.

Govindarajulu N. S., Bringsjord S. (2017). On automating the doctrine of double effect. In C. Sierra (Ed.), *Proceedings of IJCAI*, p. 4722–4730. ijcai.org.

Halpern J., Hitchcock C. (2010). *Actual Causation and the Art of Modelling In R. Dechter, H. Geffner, J. Halpern (Eds.), Heuristics, Probability, and Causality (pp. 383–406).* London: College Publications.

Halpern J. Y. (2015). A modification of the Halpern-Pearl definition of causality. In *Proceedings of IJCAI*, p. 3022–3033.

Hopkins M., Pearl J. (2007). Causality and counterfactuals in the situation calculus. *Journal of Logic and Computation*, vol. 17, n° 5, p. 939–953.

Hume D. (2012). *A treatise of human nature*. Courier Corporation.

Kant I. (1964). Groundwork of the metaphysic of morals. *New York: Harper & Row*.

Kim T.-W., Lee J., Palla R. (2009). Circumscriptive event calculus as answer set programming. In *Proceedings of IJCAI*, vol. 9, p. 823–829.

Kluckhohn C. (1951). *Values and value-orientations in the theory of action: An exploration in definition and classification*.

Kment B. (2014). *Modality and Explanatory Reasoning*. Oxford University Press.

Lifschitz V. (2008). What Is Answer Set Programming?. In *AAAI*, vol. 8, p. 1594–1597.

Lorini E. (2012). On the logical foundations of moral agency. In *DEON*, vol. 7393, p. 108–122. Springer.

Lorini E., Longin D., Mayor E. (2014). A logical analysis of responsibility attribution: emotions, individuals and collectives. *J. Log. Comput.*, vol. 24, n° 6, p. 1313–1339.

McDermott D., Ghallab M., Howe A., Knoblock C., Ram A., Veloso M. *et al*. (1998). PDDL-the planning domain definition language.

McLaren B. M. (2006). Computational models of ethical reasoning: Challenges, initial steps, and future directions. *IEEE Intelligent Systems*, vol. 21, n° 4, p. 29–37.

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

Noothigattu R., Gaikwad S. N. S., Awad E., Dsouza S., Rahwan I., Ravikumar P. *et al*. (2018). A voting-based system for ethical decision making. In S. A. McIlraith, K. Q. Weinberger (Eds.), *AAAI*. AAAI Press.

Nozick R. (1974). *Anarchy, state, and utopia.* New York: Basic Books.

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

Pereira L. M., Saptawijaya A. (2007). Modelling morality with prospective logic. In *Progress in Artificial Intelligence*, p. 99–111. Springer.

Pereira L. M., Saptawijaya A. (2017). Agent morality via counterfactuals in logic programming. In *Bridging@cogsci*, vol. 1994, p. 39–53. CEUR-WS.org.

Schiffel S., Thielscher M. (2006). Reconciling situation calculus and fluent calculus. In *AAAI*, p. 287–292. AAAI Press.

Serramia M., López-Sánchez M., Rodríguez-Aguilar J. A., Rodríguez M., Wooldridge M., Morales J. *et al*. (2018). Moral values in norm decision making. In E. André, S. Koenig, M. Dastani, G. Sukthankar (Eds.), *AAMAS*, p. 1294–1302. IFAAMAS USA / ACM.

Singer P. (2005). Ethics and intuitions. *The Journal of Ethics*, vol. 9, n° 3-4, p. 331–352.

Sloman S., Barbey A. K., Hotaling J. M. (2009). A causal model theory of the meaning of cause, enable, and prevent. *Cognitive Science*, vol. 33, n° 1, p. 21–50.

Sosa E., Tooley M. (1993). *Causation* (vol. 27) n° 1. Oxford University Press.

Struhl K., Rothenberg P. (1975). *Ethics in perspective: a reader*. Random House.

Tufiş M., Ganascia J.-G. (2015). Grafting norms onto the bdi agent model. In *A construction manual for robots' ethical systems.* Springer.

Wallach W., Allen C., Smit I. (2008). Machine morality: bottom-up and top-down approaches for modelling human moral faculties. *AI Soc.*, vol. 22, n° 4, p. 565–582.

Wu Y., Lin S. (2018). A low-cost ethics shaping approach for designing reinforcement learning agents. In S. A. McIlraith, K. Q. Weinberger (Eds.), *AAAI*. AAAI Press.