UNIVERSITÉ DE FIANARANTSOA

ECOLE DOCTORALE
MODÉLISATION - INFORMATIQUE

THÈSE DE DOCTORAT DE L'UNIVERSITÉ DE FIANARANTSOA

Domaine : Sciences et Technologies

Discipline: Informatique

Prise en compte des normes dans les comportements des agents

Présentée et soutenue par **Tokimahery RAMAROZAKA**

Le 8 Juillet 2024, à Fianarantsoa.

Président du jury:

Aimé Richard HAJALALAINA, Professeur, Université de Fianarantsoa

Directeurs :

Jean-Pierre MÜLLER, Cadre scientifique, CIRAD, UMR SENS, Montpellier.

Hasina Lalaina RAKOTONIRAINY, Maître de conférences, EMIT, Université de Fianarantsoa.

Rapporteurs :

Nicolas Raft RAZAFINDRAKOTO, Professeur Titulaire, Université de Soavinandriana-Itasy.

Laurent VERCOUTER, Professeur des Universités, LITIS, Normandie Université.

Examinateurs :

Thomas MAHATODY, Professeur Titulaire, ENI, Université de Fianarantsoa

Jean Michel BRUEL, Professeur des Universités, IRIT, Université de Toulouse.

TABLE DES MATIÈRES

I will say of the Lord, "He is my refuge and my fortress, my God, in whom I trust." (Psalm 91:2)

Remerciements

Tant de personnes à remercier pour l'accomplissement de cette incroyable épopée, et pourtant je ne pourrai pas tous les citer dans ce manuscrit pour ne pas l'allonger plus qu'il ne l'est déjà. Je remercie donc personnellement et sincèrement :

Jean-Pierre MÜLLER, mon directeur de thèse, pour m'avoir accompagné et supporté durant ces longues années, avec un optimisme (et une rigueur) sans pareil; mais également pour m'avoir appris tant de choses sur le monde de la recherche, sur moi-même. Je pourrais en dire long (très long) sur ses apports durant ces longues années, mais il faut laisser de la place pour les autres. En tout cas, vraiment : merci infiniment, Jean-Pierre !

Hasina Lalaina RAKOTONIRAINY, mon directeur de thèse également, mais aussi un ami de longue date, qui a cru en moi depuis mes débuts en Master en 2017, pour avoir toujours trouvé des solutions face à l'adversité. Un vrai modèle d'optimisme et de volonté à impacter sur le développement des jeunes, de l'éducation, et de la nation. C'est en voyant sa thèse en 2016 que j'ai eu la conviction de vouloir en faire une moi aussi, merci infiniment !

Aimé Richard HAJALALAINA, le Président de l'Université de Fianarantsoa, également un de mes premiers enseignants en Informatique, d'avoir permis l'accomplissement de ce travail et sa soutenance. Merci également pour toutes les opportunités qui nous ont été prodiguées, et qui seront encore prodiguées aux générations futures.

Mes parents et ma famille, pour leur soutien inconditionnel tant financier, moral que matériel. Merci de m'avoir toujours encouragé, aidé, conseillé et soutenu dans le calme, comme dans la tempête, dans la proximité, comme dans la distance. Merci d'avoir fait de moi qui je suis, cette thèse vous est dédiée.

Toute l'équipe de l'UMR-SENS & CIRAD, qui m'a chaleureusement accueilli, particulièrement Aurélie BOTTA, et Nathalie ROVIS, qui ont été très bienveillantes et qui m'ont permis d'intégrer dans l'UMR SENS et m'ont assisté dans l'obtention de financements pour me rendre sur place, tout comme le reste de l'équipe : Oleks, Abigail, Garance, Pierre, Léna, Sigrid, Christophe, … Merci à tous pour votre accueil.

Mes ami(e)s du CIRAD / Montpellier, qui ont partagé leurs bureaux et/ou un bout de chemin avec moi durant cette thèse, particulièrement : Monica, Mark, Sokhna, Francesca, Thibault, Mario, David, Noémie, Koloina, Fanilo, Hoby, Héloïse, Anne-Jeanne, Delphine, Viviane, Aina... Et une mention très spéciale pour Mamy Rakoto-Ravalontsalama, pour son altruisme sans pareil, pour nous avoir accueilli et guidés à chaque séjour sur Montpellier, pour tous ces moments de convivialité, de partage, de rires, de découverte : Merci Mamy !

Mes ami(e)s et proches de Madagascar, pour leur précieux soutien et leur amitié au cours de cette thèse, notamment : Alice, Sariaka, Diana, Dinah, Sitraka, Fy Tia…

Mes collègues, **étudiant(e)s, assistant(e)s de la tribu HEI** (Haute École d'Informatique), de m'avoir soutenu, de m'avoir gardé de telle sorte que je puisse concilier thèse et enseignement. Merci à Lou, Ryan, Mayah, Judicael, Vagno, Iloniavo, Jean-Marc, Sanda, Herilala, Ricka, Michou, Zoé, Sandrine, Lyvah, Vololona, Hajatiana, et toute la tribu HEI.

Les membres du jury, de mon comité de thèse et de l'École Doctorale Modélisation Informatique (EDMI), pour avoir fourni à ces travaux l'infrastructure, les compétences transversales, les retours nécessaires pour l'améliorer année après année.

Le SCAC (Service de Coopération et d'Action Culturelle) de l'Ambassade de France**,** pour m'avoir accordé deux bourses BGF (Bourse du Gouvernement Français) afin de me rendre à Montpellier dans le cadre de ce travail.

Les Sœurs de Saint-François d'Assise à Montpellier, pour leur chaleureux accueil et pour m'avoir hébergé dans leurs locaux durant mes séjours à Montpellier. Merci pour tous vos conseils, vos partages, et votre hospitalité, je ne me suis pas senti dépaysé le moins du monde.

Michael Dickens, concepteur de la disposition de clavier alternative MTGAP, avec laquelle l'entièreté de ce manuscrit a été écrite, pour m'avoir permis d'apprécier un processus de rédaction plus confortable, sans risquer de tendinites ou de syndrome du canal carpien.

…Ainsi que toutes les autres personnes qui ont contribué de loin ou de près au présent travail, durant toutes ces années: Merci !

Résumé

La modélisation à base d'agents vise à rendre compte de la complexité des comportements collectifs sur la base des interactions entre les entités. Dans le cas particulier des systèmes sociaux, les structures organisationnelles jouent un grand rôle et sont régies par des normes implicites ou explicites. Dès lors, la problématique est de comprendre comment ces normes sont modélisées et prises en compte dans les comportements individuels et structurent les comportements collectifs de façon adéquate. De nombreuses architectures ont été proposées, comme les modèles à base d'agents explorés par Müller et al., lesquels visaient à évaluer l'efficacité des normes sociales à Madagascar, mais avec des prises en compte des normes, ou des formes de normes, très simples. En réponse, cette thèse propose une architecture d'agent avec son langage dédié afin d'engendrer automatiquement ces comportements normatifs, et les rendre explicables. Nous utilisons l'Ingénierie Dirigée par les Modèles pour construire les syntaxes et la sémantique du langage, avec des extensions de la planification automatisée classique engendrant les comportements proprement dits. Pour démontrer la faisabilité des concepts proposés, un exemple de modèle est fourni. L'architecture obtenue représente un outil d'aide à la décision permettant aux juristes et autres parties prenantes de discuter, d'interpréter, et de tester différents scénarios normatifs et leurs impacts sur différentes thématiques.

Mots-clés: Planification automatisée, Systèmes Multi-Agents, Normes, Institutions, Ingénierie dirigée par les modèles.

Abstract

Agent-based modeling aims to account for the complexity of collective behavior on the basis of interactions between entities. In the particular case of social systems, organizational structures play a major role, and are governed by implicit or explicit norms. The problem is therefore to understand how these norms are modeled and taken into account in individual behavior, and how they structure collective behavior appropriately. Many architectures have been proposed, such as the agent-based models explored by Müller et al. aimed at assessing the effectiveness of social norms in Madagascar, but with very simple consideration, or representation of norms. In response, this thesis proposes an agent architecture with its dedicated language to automatically generate these normative behaviors, and make them explicable. We use Model Driven Engineering to build the syntax and semantics of the language, with extensions to classical automated planning generating the actual behaviors. To demonstrate the feasibility of the proposed concepts, an example model is provided. The resulting architecture represents a decision-support tool enabling lawyers and other stakeholders to discuss, interpret and test different normative scenarios and their impacts on different themes.

Keywords: Automated Planning, Multi-Agent Systems, Norms, Institutions, Model-Driven Engineering.

Liste des acronymes

AGR	Agent Group Role
ADICO	Attribute Deontic aIm Condition Otherwise
BDI	Belief Desire Intention (Croyances Désirs Intentions)
BOID	Belief Obligation Intention Desire
BP	Back-propagation
Cc	Contraintes de (non) codénotation
COBA	Communauté de Base
Ct	Contraintes temporelles
DIARC	Distributed, Integrated, Affect, Reflection, Cognitive
EBNF	Extended Backus-Naur Form
EDMI	Ecole Doctorale Modélisation-Informatique
EMIL-A	EMergency In the Loop Architecture
EMIT	Ecole de Management et d'Innovation Technologique
HTN	Hierarchical Task Network
IDM	Ingénierie Dirigée par les Modèles
JaCaMo	Jason Cartago Moise
GELOSE	GEstion LOcale SEcurisée
GOAP	Goal-Oriented Action Planning
LTL	Linear Temporal Logic
MOISE	Model of Organization for multI-agent SystEms
Normas	Normative Multi-Agent Systems
OC	Open Condition (condition ouverte)
PDA	Perception - Decision - Action
PDDL	Plan Domain Definition Language
POP	Partial Order Planner (Planificateur à Ordre Partiel)
PrioV	Priority of Values

RNR	Ressources Naturelles Renouvelables
RT-BDI	RealTime BDI
SMA	Systèmes Multi-Agents
SMT	Satisfiability Modulo Theories
STRIPS	STanford Research Institute Problem Solver
UCS	Uniform Cost Search
UML	Unified Modeling Language

Glossaire

ADICO	Grammaire institutionnelle populaire publiée en 1995 permettant de décrire les normes, les institutions, et les organisations.
Agent	Entité autonome représentant une entité du monde réel capable de comportements proactifs, réactifs, et sociaux pour atteindre ses objectifs.
Architecture d'agent	Architecture logicielle permettant à un agent de prendre une décision de manière autonome pour atteindre ses objectifs
Institution	Structure sociale dotée de normes et d'une ontologie pour les exprimer, à travers : des assertions, des rôles, des normes régulatrices, des normes constitutives.
HINA	Modèle à base d'agents développé en 2013 par Muller J.P. et Aubert S. à Madagascar permettant d'évaluer les impacts des normes sur la communauté locale d'Antontona
Heuristique	Fonction permettant de trouver une solution en un temps limité, même si elles sont sous-optimales. Notamment, elle permet de traiter les options les plus prometteuses en premier.
log4j2	Bibliothèque logicielle permettant de gérer des traces et des historiques d'applications.
MIRANA	Modèle à base d'agents développé en 2010 par Müller J.P., Aubert S., et al., pour évaluer les impacts des normes sur les ressources naturelles renouvelables dans la commune de Didy
Organisation	Une institution particulière où l'on spécifie quel agent joue quel rôle dans l'institution.
Plan	Séquence d'action partiellement ou totalement ordonnées permettant d'atteindre un objectif
PrioV	Ordre total de priorité sur les dix valeurs utilisées par Kammler et al., 2022
Planificateur	Programme permettant de spécifier un problème de planification, et de trouver automatiquement un plan solution si elle existe.

Liste des figures

Liste des tableaux

Liste des codes

Introduction

Rakoto a besoin de trouver à manger pour sa famille. Il va donc à la communauté de base de son village pour obtenir une licence de pêche en payant une certaine somme, avant d'aller acheter du matériel de pêche légal pour finalement se rendre dans les lieux de pêche de son village, espérant que Dame Fortune lui sourit dans sa quête de nourriture. Comme Rakoto, les normes sociales façonnent nos comportements au quotidien : dans les magasins, au travail, lors d'un match de football, sur l'autoroute etc. Ces normes prescrivent des comportements sociaux qui sont obligatoires, interdits ou permis, qui sont jugés "idéaux" pour maintenir un équilibre spécifique, mais elles peuvent être violées sous certaines circonstances: naturellement, Rakoto, peut décider de manière autonome de chasser sans licence ou de voler au marché pour atteindre ses fins, au lieu d'adopter le comportement attendu par les normes.

De ce fait, les personnes chargées de choisir et d'imposer les normes ont besoin d'évaluer leurs impacts au préalable. À plus grande échelle, il est en effet difficile pour les législateurs de proposer des normes pour assurer la durabilité des ressources naturelles renouvelables (RNR) de tout un collectif, que ce soit sur la gestion forestière, l'agriculture, la pêche : les législateurs peinent à entrevoir l'efficacité des normes à proposer.

Nous retrouvons ce questionnement, par exemple, dans le contexte de Madagascar et de son administration forestière, à partir des années 1996, lorsque la loi GELOSE (Gestion Locale Sécurisée) fut instaurée pour la durabilité des RNR [Aubert, 2002]. La loi GELOSE repose essentiellement sur un contrat de transfert de gestion à un groupement volontaire d'individus (d'un même village ou d'un même hameau par exemple), appelé les communautés de base ou COBA; ils seront chargés de gérer l'accès, la conservation, l'exploitation et la valorisation des ressources objet du transfert, notamment certaines forêts. Ce contrat confère ainsi aux COBA, un statut juridique particulier, qui leur incombe de mettre en place de règles de vie commune ou des plans d'aménagement [Sarrasin, 2009; Ramamonjisoa et al., 2012], c'est-à-dire des "normes" pour atteindre leurs objectifs communs.

Cependant, les normes issues de ces contrats de transfert de gestion eurent bien du mal à être mises en œuvre, puisque : 1) celles-ci doivent s'adapter aux situations locales en prenant en compte les motivations et les pratiques des acteurs locaux; et que 2) les ressources forestières ne peuvent systématiquement être l'objet de ce contrat de transfert entre l'État et

les COBAs. Par conséquent, les législateurs ont non seulement du mal à trouver des normes efficaces, mais en plus ils doivent le faire pour une multiplicité de territoires régies par différentes organisations. Ils sont donc en quête de réponses quant à : quelles normes proposer ? où ? quand ? et surtout, quels impacts sur la durabilité des RNR ?

Pour répondre aux questions et aux doutes des législateurs, sans s'engager dans des coûts exorbitants en se servant de la population réelle comme des objets de test, des modèles de simulation informatique sont de plus en plus explorés à la place. Pour le cas de Madagascar, des modèles, dits, des modèles à base d'agents ont déjà été entrepris : le modèle MIRANA [Aubert et al., 2010] et le modèle HINA [Aubert & Müller, 2013], développés par Müller et al. sur la plateforme MIMOSA [Müller, 2004]. Ces modèles ont pour but d'évaluer l'efficacité des normes (e.g : reboisement, collectes de taxes..) que les COBA se devaient de prôner, afin de démontrer leurs impacts sociaux, économiques et environnementaux à la population locale, en utilisant des entités virtuelles dits "agents" représentant les gardes forestiers, les foyers, et les autres entités du système avec leurs comportements. Afin d'avoir en perspective les caractéristiques des modèles à base d'agents, la figure 1 présente un exemple visuel sur la plateforme MIMOSA.

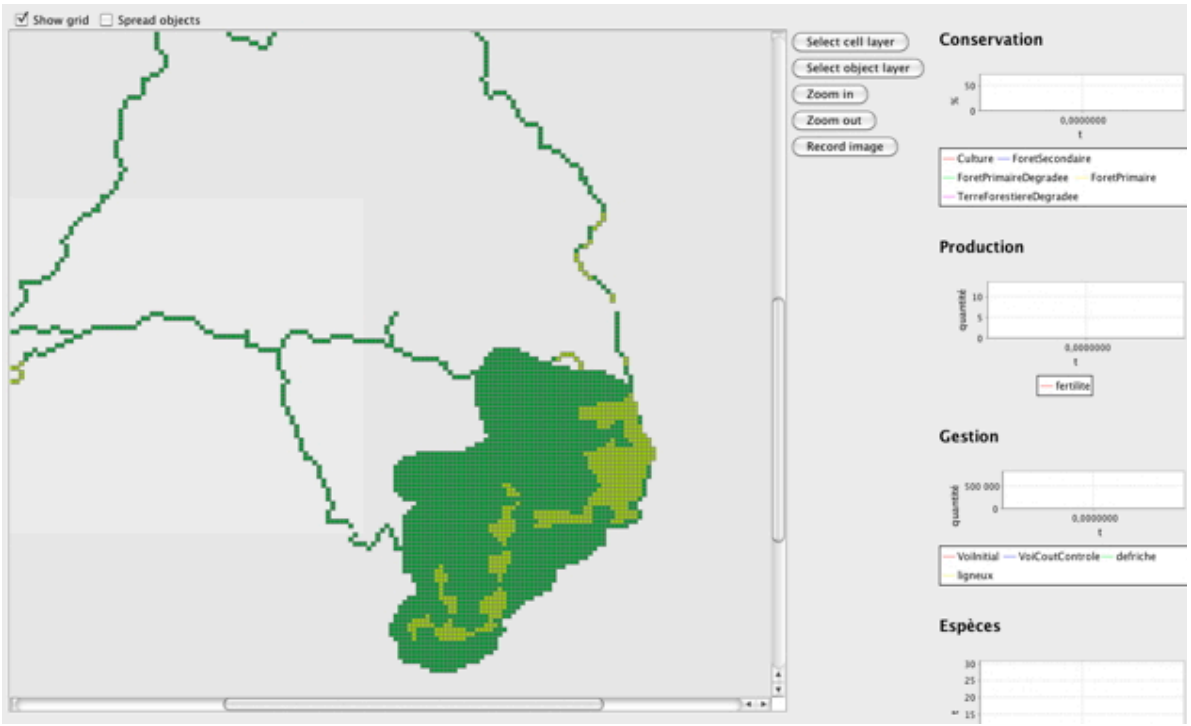


Figure 1: Un exemple de simulation avec MIMOSA avec les indicateurs observés à droite (source : mimosa.sourceforge.net/documentation.html)

L'avantage de ces modèles est de pouvoir tester différents scénarios, avec des normes paramétrables, et en mesurer les impacts sur le comportement des acteurs, représentés par des "agents", et le socio-écosystème en général. À terme, les interprétations de ces modèles en font un outil d'aide à la décision et la discussion, car les parties prenantes (agriculteurs, écologues, villageois…) peuvent y intégrer leur point de vue, et décider plus efficacement.

Si l'utilité des modèles à base d'agents dans la compréhension de systèmes complexes, telle la gestion des RNR à Madagascar n'est plus à prouver, il n'en est rien pour ce qui est de la praticité des outils capables de les construire. Les deux modèles MIRANA et HINA, tous deux conçus sur l'architecture FARITRA [Rakotonirainy, 2016] de MIMOSA représentent et traitent les normes au cas par cas, de manière ad-hoc, selon [Aubert et al., 2010]. Plutôt que d'offrir une structure bien définie des normes au modélisateur pour articuler les normes, elle est décrite avec un langage informatique selon ce que décide l'informaticien, ce qui rend l'exploration des résultats difficile, et réduit les apports des résultats obtenus par rapport aux questions des juristes et des agronomes. Pour leur prise en compte, MIRANA propose une prise en compte assez simplifiée :

- Chaque foyer dans le modèle planifie ses actions en fonction de ses besoins, et cherche un contexte dans lequel les actions pour les satisfaire sont autorisées;
- S'il n'en trouve aucune, il lâche prise de certaines normes en priorité;
- Il demande ensuite une licence aux autorités locales. Si cette licence n'est pas accordée, l'action devient illégale ou n'est pas effectuée.

Ce processus pose problème, dans la mesure où 1) la modélisation des normes et les détails de leur prise en compte sont non seulement limités en expressivité, mais également en explicabilité, et 2) au vu de la pluridisciplinarité des thématiques abordées par les juristes, les économistes, et les autres spécialistes, on ne s'accorde pas forcément sur "où, quand et à qui ?" une norme est appliquée. Ayant besoin d'un outil pour spécifier leurs normes, leur aspect spatial, temporel, et social, il est important de pouvoir fournir une spécification plus formelle, et de l'implémenter pour en faire un réel outil d'aide à la décision et à la discussion.

Pour illustrer ces problèmes, prenons par exemple la norme suivante : "il est interdit de pêcher en hiver, autour des lacs protégés". Hormis le fait qu'il faut écrire le code informatique nécessaire pour décrire la norme et son contexte spatio-temporel, l'architecture reste floue quant à comment les acteur(s) de ces modèles en déduisent qu'il faut attendre la prochaine saison, ou pêcher dans des lacs non protégés pour se conformer à cette norme. L'absence de spécification sur les aspects spatiaux (où une norme est-elle applicable ?),

temporels (quand est-elle applicable ?) et sociaux (à qui s'applique-t-elle ?), et leurs impacts, amenuisent la capacité des modèles à répondre aux questions des décideurs sur l'efficacité des normes proposées.

Dans la littérature sur les modèles à base d'agents, malgré la présence de plusieurs travaux sur la théorie des normes, et de leur prise en compte dans le raisonnement des agents, force est de constater que :

- Plusieurs aspects sur la prise en compte des normes laissent à désirer : notamment les stratégies utilisées par les agents en réponse à ces normes dans leur comportement;
- Plusieurs approches limitent les comportements des agents par un ensemble de comportements prédéfinis alors qu'il est difficile de prédéfinir tous les comportements possibles face aux normes, surtout dans des environnements ouverts.

Face à cette problématique de la prise en compte des normes peu expressives dans les modèles à base d'agents, le présent travail propose comme solution une architecture d'agent et son langage dédié. Il ambitionne de fournir une représentation formelle des normes, mais aussi une spécification de comment le comportement de l'agent est influencé par ces dernières, et d'en faire un outil permettant aux juristes ou autres spécialistes du domaine d'évaluer les impacts de différents scénarios de normes.

En utilisant l'ingénierie dirigée par les modèles (IDM) pour construire la syntaxe abstraite, la syntaxe concrète et la sémantique opérationnelle du langage, l'objectif de ce travail est de fournir un outil permettant de tester comment différentes normes affectent le comportement de l'agent et l'ensemble du socio-écosystème. Un modèle à titre de preuve de concept est également proposé pour démontrer le type de problème que l'outil développé peut traiter, afin de démontrer in fine le fait qu'il soit généralisable sur d'autres thématiques, en dehors de la gestion des RNR.

Pour situer notre contribution, [Conte & Castelfranchi, 1999] décrit la prise en compte des normes en deux étapes bien distinctes : 1) la reconnaissance des normes qui s'appliquent à l'agent, et de 2) la prise en compte d'une norme : prendre une décision en prenant en compte ces normes applicables dans ses décisions. Nos contributions se situent au niveau du deuxième point en poursuivant les travaux de [Raharivelo & Muller, 2018], qui permettent d'ores et déjà de calculer toutes les normes applicables avec un contexte spatio-temporel, et social.

Le présent travail se distingue de ces travaux antérieurs, en se concentrant sur les différentes stratégies par lesquelles l'agent peut générer un comportement adapté aux normes. Le reste de ce document sera divisé comme suit :

- La première section fournit un état de l'art pour revoir les normes, la modélisation à base d'agent, et les méthodes actuelles pour prendre en compte les normes, ainsi que leurs limites respectives.
- La section 2 présente et justifie les méthodes utilisées pour nos contributions;
- Puis, la section 3 décrit les résultats obtenus.
- La section 4 discute de leur pertinence par rapport à la problématique traitée.
- La section 5 relate des différentes perspectives pour de futurs travaux
- Enfin, la section 6 conclut nos travaux de recherche en donnant une vue d'ensemble de ce qui a été effectué, en rappelant l'adéquation des contributions aux objectifs posés.

1. État de l'art

Pour pouvoir analyser les impacts des normes dans les modèles à base d'agents, et fournir un outil de modélisation plus expressif aux juristes, nous commençons par expliciter le concept de normes, et in fine, leur lien avec les modèles à base d'agents, plus spécifiquement comment elles impactent sur le comportement produit par l'agent.

Au terme de cet état de l'art, il sera possible de déduire les apports et les limites des travaux existants dans le domaine étudié, dans l'optique de mettre en exergue non seulement la pertinence de ce travail, mais aussi pour justifier le continuum des choix adoptés lors de sa conception dans les sections suivantes. L'état de l'art présentera successivement dans l'ordre :

1. Les normes : pour définir et formaliser ce qu'est une norme et ce qu'elle implique au niveau des comportements au sein d'un groupe ou d'une société, et ses liens avec les modèles à base d'agents;
2. Les modèles à base d'agents : pour comprendre ce qu'est un agent, un modèle à base d'agents et ses caractéristiques;
3. Les architectures d'agents : pour comprendre comment les comportements autonomes des agents sont engendrés ou choisis (sans les normes);
4. Les architectures d'agent normatives : pour retracer comment les normes ont été prises en compte dans les architectures d'agent dans la littérature;
5. L'implémentation des normes dans les SMA: pour comprendre comment les différents langages et outils dits normatifs intègrent les normes dans les modèles à base d'agents;
6. La planification automatisée: pour descendre à un niveau de détail plus fin sur comment un comportement d'agent peut être créé de zéro;
7. Les approches en termes de planification automatisée normative pour revoir comment les normes ont été intégrées à l'engendrement de comportements.

Au terme de cet état de l'art, nos contributions par rapport aux constats effectués sont présentées, suivi des résultats obtenus après leur implémentation.

1.1. Les normes

Selon [Crawford & Ostrom, 1995] : les normes (également dites “normes sociales”) sont décrites comme étant une compréhension partagée des actions qui sont obligatoires, interdites, ou permises.

Les normes représentent des règles et des standards qui sont prises en compte par un groupe d’individus, et qui guident ou restreignent leur comportement social sans l’appui de la loi [Cialdini & Trost, 1998]. L’utilité des normes est par conséquent de dresser un comportement idéal attendu au sein d’un groupe, d’une communauté ou d’une société afin de favoriser la coordination et réduire les frictions sociales, selon [Young, 2007]. Elles représentent donc une certaine régularité, qui peut émerger des interactions entre les membres de l'organisation au fil du temps, ou qui est délibérément imposée par une autorité.

Face à la présence des normes, chaque individu concerné peut agir différemment dans une même situation, en fonction de la valeur qu’ils accordent à la prise en compte (ou à la violation) des normes [Ostrom, 2009]. Deux des caractéristiques clés des normes sont la possibilité d’une violation et leur évolutivité, c’est-à-dire :

1) Les normes peuvent être violées : vu les potentiels conflits entre les normes et les objectifs personnels de chaque individu, mais également entre les normes elles-mêmes qui forcent souvent à respecter une norme au détriment de l’autre si les normes dans le pire des cas;
2) Les normes peuvent évoluer : une norme peut émerger des interactions individuelles [Ulmann-Margalit, 1977] et peut éventuellement être innovée ou abolie [Conte & Castelfranchi, 1995; Conte & Castelfranchi, 2006];

Les recherches sur les normes ont pour objectif d’analyser leur conception, ainsi que leurs effets sur le comportement individuel des individus, par exemple dans la gouvernance des biens communs [Ostrom, 1990], pour trouver quel ensemble de normes permettrait d’atteindre un objectif donné. Pour illustrer l’intérêt des normes dans des thématiques sociales, environnementales et économiques, nous allons citer, entre autres :

1) [Novo & Garrido, 2014] qui évalue les impacts de l’implémentation des nouvelles lois sur l’eau au Nicaragua ;

2) [Aubert et al., 2010; Aubert & Müller, 2013] qui évalue l'efficacité des systèmes régulatifs que les communautés de base doivent mettre en place à Madagascar pour atteindre une durabilité des ressources naturelles renouvelables, sur le plan environnemental, économique, et social;
3) [Carter et al., 2015] qui évalue la divergence entre des lois et les régulations locales existantes dans le domaine de l'alimentation biologique;
4) [Smajgl et al, 2008; Frantz et al., 2015; Ghorbani & Bravo, 2016] qui démontrent et analysent l'émergence de comportements réguliers dans la gestion de ressources communes;
5) [Dignum, 2021] qui teste les effets des applications de traçage des cas de COVID-19 sur l'évolution des infections, que ce soit par contact, dans les lieux publics ou dans les foyers, avec différents taux d'utilisation de ces applications.

Pour comprendre les tenants et aboutissants de ce type de travail, la suite de cette section sur les normes se déroule comme suit : Premièrement, nous allons énumérer les différents types de normes; deuxièmement, les notions d'institutions et d'organisations seront présentées; troisièmement, nous allons présenter les représentations formelles de ces entités avant de déboucher sur une synthèse de l'importance des normes et des défis potentiels dans leur mise en œuvre.

1.1.1. Typologie des normes

On distingue deux types de normes selon [Coleman, 1998] en fonction de leur utilité : les normes essentielles et les conventions.

1) Les normes essentielles qui servent à régler les conflits entre intérêts personnels et intérêts communs. Par exemple, la norme essentielle qui décrit qu'il est interdit de polluer les routes peut inciter une personne à jeter ses ordures à la poubelle, plutôt que sur le trottoir.
2) Les conventions qui émergent naturellement sans aucune imposition quand il n'y a pas conflit entre les intérêts individuels et collectifs. Ce sont des comportements habituels, attendus où tout le monde s'y conforme et s'attend à ce que les autres s'y

conforment également; un exemple de convention serait le fait qu'il est obligatoire de dire *«bonjour»*.

Les normes sont donc des dynamiques sociales qui peuvent être formelles, comme le cas des normes essentielles, ou informelles, comme le cas des conventions, et qui peuvent également coexister dans un même espace. Pour le cas des normes essentielles, elles vont se décliner en trois types :

a) les normes régulatrices : qui décrivent un comportement idéal à l'aide d'une modalité déontique exprimant une obligation, une interdiction, ou une permission [Ostrom & Crawford, 1995];
b) les normes constitutives : qui expriment qu'un objet ou un individu compte comme un autre concept partagé d'une organisation [Searle, 1969];
c) les normes procédurales : qui expriment comment les normes doivent être mises en place ou comment une certaine action doit être exécutée, par exemple, en disant qu'une administration doit se composer à 50% de femmes et 50% d'hommes, ou qu'il faut pêcher avec tel outil [Lawrence, 1976].

Les conventions, tout comme les normes régulatrices peuvent s'exprimer grâce à une modalité déontique : obligation, interdiction, et permission. Vu que l'orientation du présent travail se focalise sur la prise en compte des normes et non sur les motifs sous-jacents à leur création et à leur émergence, nous allons utiliser le terme "norme régulatrice" pour englober à la fois les conventions et les normes régulatrices, sans établir de distinction entre elles.

Pour illustrer à l'aide d'exemples, une norme régulatrice pourrait décrire qu'il est obligatoire pour tout villageois d'avoir une licence pour pouvoir chasser dans une forêt protégée, tandis qu'une norme constitutive permettrait de générer des inférences en stipulant qu'un individu *Paul* compte comme un villageois, et que la forêt de *Ranomafana* compte comme une forêt protégée, et donc, par inférence, *Paul* doit avoir une licence pour chasser dans la forêt de *Ranomafana* (et potentiellement tout autre forêt protégé). Pour résumer, le diagramme de classe UML (Unified Modeling Language) de la Figure 2 retrace l'ensemble des différents types de normes selon cette classification.

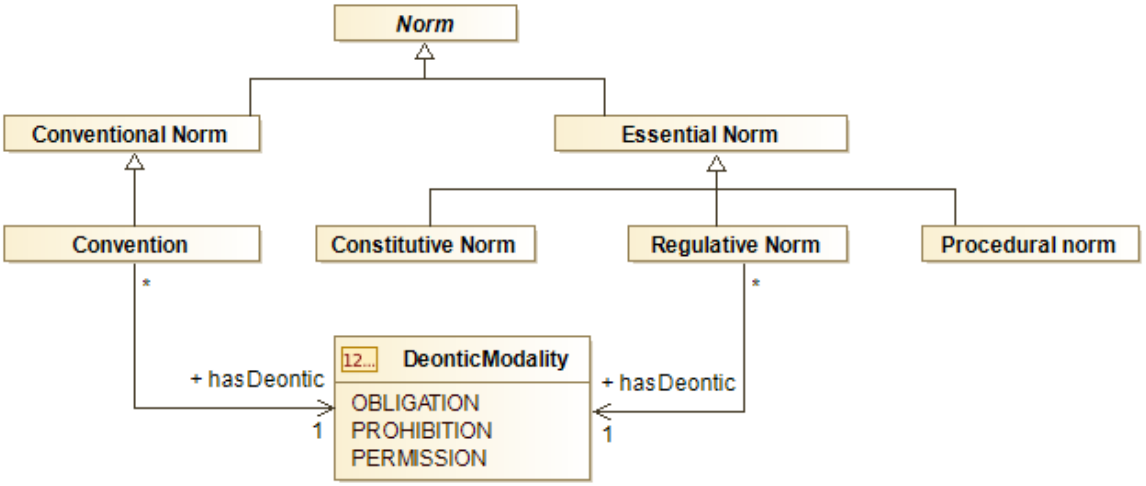


Figure 2: Typologie des normes, adapté de [Mahmoud et al., 2014]

1.1.2. Les institutions et les organisations

Puisque l'objectif des normes est de réguler les comportements au sein d'un groupe, d'une société, ou d'une communauté, il nous faut décrire les structures, qui décrivent les normes et de quelle manière ? pour quel groupe de personnes ? À terme, l'objectif est de déduire leurs implications pour les membres du groupe en question. En fonction des auteurs, le terme "institution", "organisation", et/ou de "groupes" est utilisé.

Dans le cadre de cette polysémie, [Ostrom, 2019] mentionne qu'une des difficultés pour pouvoir analyser les institutions est le fait que le terme "institution" peut faire référence à plusieurs entités qui sont soit :

a) les organisations elles-mêmes (e.g. : la commune de *Didy*, la forêt de *Ranomafana*, etc.) ou ,
b) l'ensemble des normes qui sont utilisées par des individus qui agissent à travers ces organisations (e.g : tout membre de la commune de *Didy* doit payer des impôts, il est interdit de chasser dans telle partie de la forêt de *Ranomafana*).

Pour comprendre comment les normes sont reliées aux notions d'institutions et d'organisations, et les différencier, nous allons revoir les définitions accordées à ces deux concepts.

1.1.2.1. Les institutions

La notion d'institution désigne le plus souvent : a) des structures aux comportements stables et réguliers, potentiellement émergentes et uniquement visibles du point de vue d'un observateur externe [Ferber, 1994] ou b) un ensemble de normes ou de règles d'une société qui vont décrire ce qui est obligatoire, permis, ou interdit pour certains individus, ou un groupe d'individus [Searle, 2005].

Pour illustrer, l'adhésion à ces deux concepts : [Ostrom, 1990] définit une institution comme étant l'ensemble des règles utilisées par un ensemble d'acteurs pour organiser des activités répétitives qui auront des retombées sur ces mêmes acteurs, voire potentiellement d'autres acteurs, avec un mécanisme de renforcement. Cette définition converge vers celle de [North, 1990], qui décrit les institutions comme étant les règles du jeu dans une société, et des contraintes (formelles ou informelles) pour structurer les interactions politiques, économiques, et sociales. Ces normes dites "normes régulatrices" se présentent sous la forme d'une condition "**Si** *condition(s)* **alors** *il est* interdit, permis ou obligatoire *de faire une certaine action ou de maintenir un certain état*". Par exemple :

- "**Si** *une rivière compte comme un territoire sacré*, **alors** *il est interdit d'y pêcher*."
- "**Si** *un individu est chef de famille*, **alors** *il est obligatoire pour cet individu de toujours avoir à manger pour sa famille*".

À contrario, [Searle, 2005] dans son ouvrage intitulé *"What is an institution ?"*[1], décrit une institution comme étant tout système de règles accepté, contenant des règles constitutives de la forme *"X compte comme Y"*, *X* peut être un objet ou un individu du système, et *Y* est un concept existant de l'institution, i.e. un rôle. Par conséquent, une institution est non seulement une structure composée de normes régulatrices de la forme *"Si condition alors il est obligatoire / permis / interdit de faire / maintenir X"*, mais aussi

[1] Signifie littéralement "Qu'est-ce qu'une institution ?".

composée de normes constitutives rattachant un objet / individu à un rôle de l'institution, pour pouvoir déduire les normes applicables à un individu. Par exemple, une institution pose les normes suivantes :

- Norme régulatrice : "**Si** un villageois n'a pas de licence **alors** il lui est interdit de pêcher dans les lacs protégés";
- Normes constitutives : "*Toky* **compte comme** un villageois", "*Tritriva* **compte comme** un lac protégé";

À travers le fait que individus du système tel que *Toky*, et les objets tels que *Tritriva* jouent les rôles de "*Villageois*" et de "*Lac protégé*" respectivement, ces normes permettent d'inférer que "**Si** *Toky n'a pas de licence* **alors** *il lui est interdit de pêcher dans le lac Tritriva (ainsi que les autres lacs protégés)*".

1.1.2.2. Les organisations

La notion d'organisation, décrit en sciences sociales *"comment"* s'organisent les membres d'une société : elle représente l'ensemble des manières pour diviser le travail en tâches distinctes et leur coordination [Mintzberg, 1989].

[North, 1991] qui a décrit clairement les institutions comme étant les "règles du jeu", définit abstraitement les organisations comme étant les joueurs qui obéissent à ces règles [North, 1994]. Il illustre l'organisation en distinguant la structure abstraite du collectif (l'institution) des individus réels qui prennent part au collectif (l'organisation), mais les commentaires soulevés par [Hodgson, 2006] permettront de compléter qu'une organisation est un type particulier d'institution, qui peut aussi avoir ses propres normes en plus d'être un ensemble d'acteurs jouant des rôles dans une institution.

Cette idée de considérer les organisations comme un type spécial d'institution, est également retrouvée dans le cadre de la modélisation à base d'agents de [Ferber et al., 2003] à travers le modèle AGR (*Agent-Group-Role*) avec d'autres terminologies. Il définit que les agents jouent un/des rôle(s) dans des groupes, un groupe étant défini lui-même par une structure : les structures organisationnelles. Ferber trace donc la différence entre une institution et une organisation avec deux entités :

- les structures organisationnelles (=les institutions), comme étant la structure abstraite d'une organisation qui spécifie 1) la partition des groupes et les relations entre groupes, 2) la structuration des rôles par groupe et les relations entre rôles, 3) les modalités de gestion des rôles et leur lien avec le comportement des agents i.e. des individus;
- les organisations concrètes, comme une mise en œuvre (ou une instanciation) possible d'une structure organisationnelle, où des entités vont prendre effectivement part aux structures organisationnelles, que Ferber désigne comme étant un "groupe".

Dans le cadre de la modélisation à base d'agent [Hannoun et al., 1999] décrivent le modèle organisationnel MOISE (Model of Organization for multI-agent SystEms), où les organisations sont vues comme étant un moyen de répartir les tâches entre les individus et de fixer leurs relations.

Par rapport aux institutions, il ressort de ces définitions que l'organisation représente la mise en relation des rôles d'une institution avec des agents. Comme affirmé dans [Hodgson, 2006], les organisations sont un type spécial d'institution avec potentiellement ses normes en plus, mais qui précise quel agent joue quel rôle dans une méta-institution qui lui est associée.

1.1.2.3. Synthèse sur les institutions et les organisations

Au vu des quelques définitions d'une institution et d'une organisation, nous pouvons déduire que les terminologies varient selon les auteurs : les institutions de [Searle, 2005] par exemple, correspondent aux organisations de [North, 1990]; les groupes de Ferber correspondent à des entités organisationnelles chez [Hannoun et al., 1999]. Cependant, ces concepts dénotent tous vers deux concepts clés :

1. La structure : Un concept pour spécifier la structure d'une entité sociale pourvue de normes et de rôles, ainsi que les relations entre les différents rôles;
2. L'instanciation: Un concept pour créer une instance de cette structure en organisant des agents concrets aux rôles de la structure, et en ajoutant potentiellement des normes additionnelles.

Nous retrouvons donc les correspondances de cette structuration et de cette instanciation à travers le Tableau 1 [Ostrom, 1990; North, 1990; Ferber, 2003; Searle, 2005; Hannoun et al., 1999].

Tableau 1: Comparatif des concepts de structuration et d'instanciation

Concept	**Ostrom**	**North**	**Ferber**	**Searle**	**Hannoun et al**
Structure	(Auto-décrite par les normes existantes, ou à l'aide de documents)	Institution	Structure organisationnelle	Structures institutionnelles (auto-décrites par les institutions)	Structure organisationnelle (So)
Instanciation	Institutions	Organisation	Groupe (ou organisation concrète)	Institution	Entité organisationnelle (Eo)

Pour comprendre les impacts des normes dans les comportements des agents qui nous intéressent, il faut préalablement définir "comment" ces normes sont décrites par les institutions, et comment elles sont instanciées en tant qu'organisation. Nous choisissons d'utiliser de ce fait, le terme *institution* pour désigner la structure sociale, et le terme *organisation* pour désigner son instanciation.

En termes de structure interne, une institution est composée d'un ensemble de normes et d'une ontologie pour pouvoir décrire ces normes et d'autres connaissances partagées [Ostrom, 2005a]. Nous définissions dans les sections suivantes ce qu'est une ontologie.

Définition (*ontologie*) : *Une ontologie est un ensemble de terminologies qui définit les normes d'une institution à l'aide de méta-concepts, de concepts, et des assertions qui permettent d'associer des identifiants à ces concepts*.

L'ontologie constitue les mots utilisés par une institution pour décrire l'application des normes, avec :

- Les concepts qui décrivent les mots au sein d'une institution. Par exemple, pour le cas d'une communauté villageoise, nous aurons le concept de pêcher, de zone, de poissons, de membre du village, de chef du village, et d'endroits sacrés;

- Les méta-concepts qui décrivent les concepts et leur domaine, i.e. le type d'objet du concept, ainsi que le type d'information valide qu'on peut y rattacher. Des exemples basiques de méta-concepts sont le concept de "*pêcher*" qui a comme méta-concept "*action*" puisque pêcher est un concept d'action; des exemples similaires sont : 1) le concept de "*zone*" a comme méta-concept "*espace*", ou encore le concept de "*poisson*" qui a comme méta-concept "*resource*";
- Les assertions associent des noms aux concepts : elles permettent de définir quel mot est associé à un concept d'individu, de catégorie ou de relation.

Une institution doit obligatoirement disposer d'une ontologie pour décrire ce que [Crawford & Ostrom, 1995] appellent des faits institutionnels, qui sont les normes elles-mêmes. L'idée est de décrire une institution à l'aide d'une ou de plusieurs normes pour décrire ce qui y est permis, interdit, ou obligatoire pour un ensemble d'agents dans un contexte spatial, temporel, et/ou social particulier.

1.1.3. Représentation des institutions

L'outil le plus utilisé pour analyser et représenter les normes dans la littérature est la syntaxe d'une grammaire institutionnelle de [Crawford & Ostrom, 1995] nommée ADICO (Attribute Deontic aIm Condition Or else) représentée sur la Figure 3.

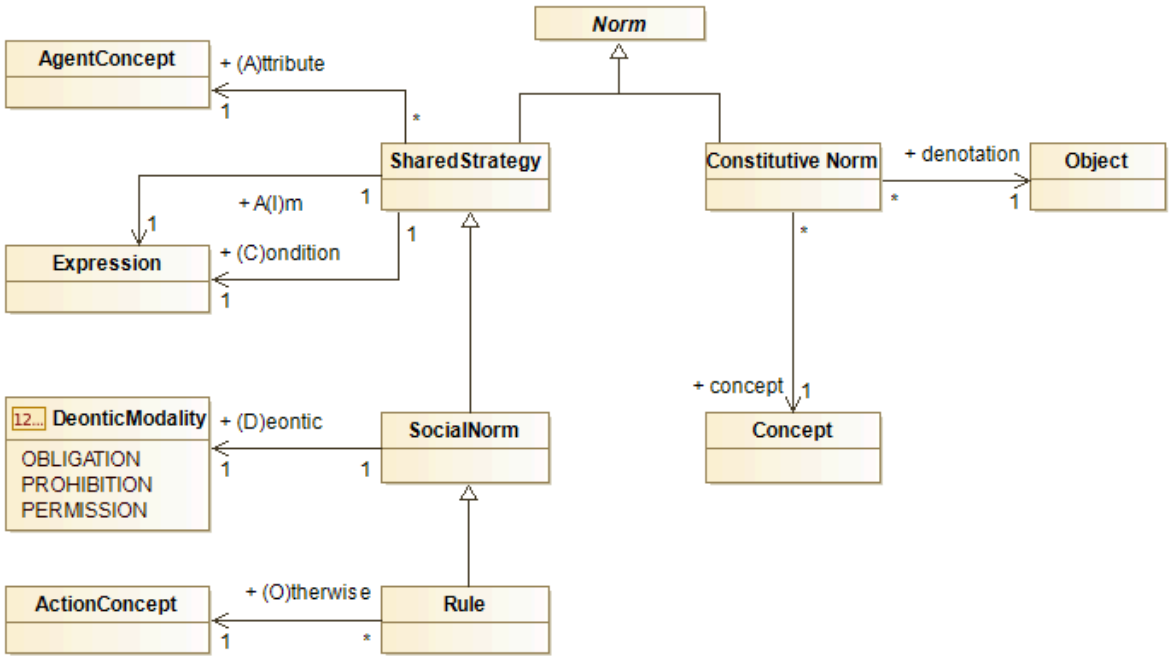


Figure 3: Représentation en diagramme de classe de la grammaire institutionnelle ADICO

Celle-ci présente une catégorisation plus fine des normes régulatrices à travers les stratégies partagées, les normes sociales, et les règles. La grammaire institutionnelle ADICO décrit les différents composants d'une norme :

- la partie *A* spécifie *qui* sont les individus / objets concernés par la norme;
- la partie D spécifie la modalité déontique indiquant s'il s'agit d'une obligation, d'une interdiction, ou d'une permission;
- la partie I spécifie les activités ou l'état des choses prescrit par la norme. Il est généralement représenté sous la forme d'un verbe;
- la partie C spécifie les conditions d'applicabilité de la norme, c'est-à-dire dans quel contexte spatial, temporel ou social est-elle applicable ?;
- la partie O spécifie les sanctions et récompenses associées respectivement à la violation et à la prise en compte de la norme.

La syntaxe ADICO décrit une norme régulatrice comme étant une stratégie partagée lorsque celle-ci dispose de la partie A, I, et C uniquement (AIC). Lorsqu'on ajoute la modalité déontique de la partie D, celle-ci devient une norme sociale (ADIC). Et lorsqu'il y a une récompense et/ou une sanction à la norme avec la partie O, la norme devient une règle (ADICO).

Par exemple, pour une même norme sur des villageois nous aurions les formulations suivantes :

- en tant que stratégie partagée : si les villageois coupent dans la forêt, ils déclarent leur coupe à la commune;
- en tant que norme sociale : si les villageois coupent dans la forêt, alors il est obligatoire qu'ils déclarent leur coupe à la commune;
- en tant que règle : si les villageois coupent dans la forêt, alors il est obligatoire qu'ils déclarent leur coupe, sinon il faudra payer une amende.

1.1.4. Synthèse sur les normes

Pour conclure, les normes sont des contraintes qui guident ou restreignent le comportement d'un ensemble d'individus pour atteindre un objectif commun. Elles sont indissociables de la notion d'institution qui représente la structure qui les a émises et qui les décrit à l'aide d'un ensemble de normes régulatrices et de normes constitutives.

Ces normes sont décrites grâce à une ontologie qui représente formellement le vocabulaire utilisé par l'institution, [Crawford & Ostrom, 1995] propose la syntaxe d'une grammaire institutionnelle appelée ADICO qui constitue une des grammaires les plus utilisées et étendues actuellement [Frantz & Siddiki, 2020; Siddiki & Frantz, 2023]. Pour pouvoir associer un groupe d'individus aux rôles décrits dans les institutions, nous utiliserons les normes constitutives pour pouvoir associer les objets / agents du système aux rôles décrits par les concepts de l'institution : c'est le concept d'organisation.

Si cette grammaire institutionnelle permet de faire des analyses institutionnelles dans divers domaines pour comprendre le rôle des institutions vis-à-vis de l'économie, de la gestion des biens communs, ou de la durabilité des biens communs, elle présente également quelques défauts.

Tout d'abord, la popularité de cette grammaire institutionnelle peut s'expliquer par le fait que : i) elle permet d'analyser et définir une entité sociale d'une manière proche des énoncés linguistiques (sujet, verbe, complément par exemple), et ce faisant elle permet de relever facilement les nuances entre des institutions similaires; par conséquent ii) elle offre un outil de synthèse pour discuter des théories qui étaient difficilement comparables auparavant, comme les approches évolutives de [Axelrod, 1986] (sur l'émergence de comportements coopératifs), et les normes sociales de [Ullmann-Margalit, 1977] (sur les impacts des normes sur le comportement des individus);

Parmi les limites d'ADICO, nous pouvons noter la difficulté de l'identification des normes. Par exemple, les travaux de [Watkins & Westphal, 2016] confirment cette difficulté en affirmant que les individus enquêtés sur les normes "ne parlent pas avec des faits institutionnels"[2], d'où les travaux évolutifs qui extraient des normes à partir des comportements émergents à travers des modèles à base d'agents [Frantz, 2015].

Quant aux lacunes des ontologies mises à disposition, le manque d'expressivité sur le temps et l'espace dans les normes est mentionné dans [Raharivelo & Muller, 2018], c'est-à-dire qu'il est difficile d'expliciter des lieux et des intervalles de temps dans les faits institutionnels.

Par exemple, si une norme en ADICO décrit que : "*Pour tous les non-résidents dans la région de Fianarantsoa, s'ils collectent des animaux sauvages dans des lieux publics, alors il est obligatoire d'avoir une autorisation, sinon, ils seront emprisonnés à la prison d'Ankazondrano et paieront une amende*", l'ontologie ne dispose pas de concepts et de méta-concepts dédiés pour le temps et l'espace, or que ce dernier est utilisé dans les composants A-I-C-O. Il faut donc se donner les moyens de les interpréter pour calculer si : 1) une est applicable à un agent ? 2) comment elle impacte son comportement? et 2) comment elle impacte l'application et la vérification des sanctions ?

En réponse, une proposition pour intégrer le contexte spatial, temporel et social dans les conditions d'une norme est proposée par les travaux de [Raharivelo & Muller, 2018; Raharivelo & Muller, 2021] pour calculer leur applicabilité avec un langage dédié, mais ne décrit pas comment les agents produisent du comportement en conséquence. L'aspect temporel est également intégré par les travaux de [Figueiredo et al., 2010] dans le langage de modélisation normatif NormML, qui propose de définir des intervalles de temps dans les conditions d'applicabilité des normes.

Une autre limite d'ADICO et de l'analyse institutionnelle en général est le manque d'explicitation sur le lien entre les normes et les comportements des individus [Siddiki et al., 2022]. Si les faits institutionnels permettent de décrire les normes, les règles, et les stratégies, il reste à comprendre les motifs qui poussent les individus à les prendre en compte ou à les violer [Crawford & Ostrom, 1995, p. 583].

Le défi de l'application future des grammaires institutionnelles serait de comprendre comment des modifications sur la conceptualisation des institutions impacterait les comportements des individus, i.e. comment les normes sont interprétées, transformées, et

[2] "People don't talk in institutional statements" (Watkins et Westphal, 2016)

prises en compte dans le comportement des individus ? Si les normes décrivent souvent ce qu'il faudrait faire, très peu de travaux s'intéressent à "*comment*" le faire.

Quelques solutions suggérées par [Siddiki et al., 2022] pour comprendre la prise en compte des normes par les individus seraient d'utiliser des modèles à base d'agents comme [Smajgl et al., 2008; Ghorbani et al., 2013; Ghorbani et al, 2015] et la théorie des jeux comme [Ostrom, 2005b]. Pour cela, il est nécessaire de développer des approches pour automatiser le codage des faits institutionnels pour les rendre moins ardus, plus génériques, et plus robustes face à de grands volumes de données institutionnelles [Siddiki et al., 2022]. C'est dans ce continuum de "comment" sont prises en compte les normes exprimées sous forme de faits institutionnels que nous introduisons dans 1.2 : Les modèles à base d'agents.

1.2. Les modèles à base d'agents

Les modèles à base d'agents[3] sont des modèles de simulation informatique où l'on s'intéresse aux interactions entre des agents autonomes, situés dans un environnement, dans l'objectif de pouvoir répondre aux questions posées par les thématiciens.

1.2.1. Les agents

Les agents peuvent tout aussi bien représenter un individu au comportement complexe (un villageois, un pêcheur, un animal…) ou une entité plus simple au comportement simple (la biomasse, l'herbe, la forêt, le feu…). Ces modèles sont construits à l'aide d'agents individuels, dont les comportements et interactions vont faire émerger les dynamiques du système.

Définition 1 (agent) : *Un agent est un système informatique situé dans un certain environnement et capable d'agir de manière autonome dans cet environnement afin d'atteindre ses objectifs.* [Wooldridge & Jennings, 1995].

[3] également appelé "les systèmes multi-agents" de manière interchangeable

Nous parlons également de systèmes multi-agents lorsqu'il y a plusieurs agents au sein d'un même environnement. Pour chaque agent, [Wooldridge & Jennings, 1995] dénombre quatre (04) caractéristiques distinctes :

- Un agent est *autonome*, il peut organiser ses actions vis-à-vis des objectifs qui lui sont propres;
- Un agent est *social*, il a la capacité d'interagir avec d'autres agents;
- Un agent est *réactif*, il peut percevoir les événements externes et y réagir en conséquence;
- Un agent est *proactif*, il est capable de produire un comportement pour satisfaire en fonction de ses objectifs et de ses autres composants internes.

La modélisation à base d'agents montre des résultats prometteurs dans des thématiques impliquant des institutions telles que 1) l'évaluation des impacts socio-économiques des normes dans la gestion forestière de Madagascar [Aubert et al., 2010; Aubert & Muller, 2013]; 2) une discussion sur les impacts d'engrais alternatifs par rapport aux pâturages et aux fermiers [Ding et al., 2015], ou encore plus récemment, l'étude de l'empreinte carbone liée à un plan de gestion de ressources naturelles renouvelables [Christensen et al., 2020]. Nous pouvons encapsuler ces définitions dans la Figure 4 où l'agent perçoit son environnement en entrée, et agit sur son environnement dans lequel il est situé, en faisant preuve de comportements réactifs, proactifs et/ou sociaux.

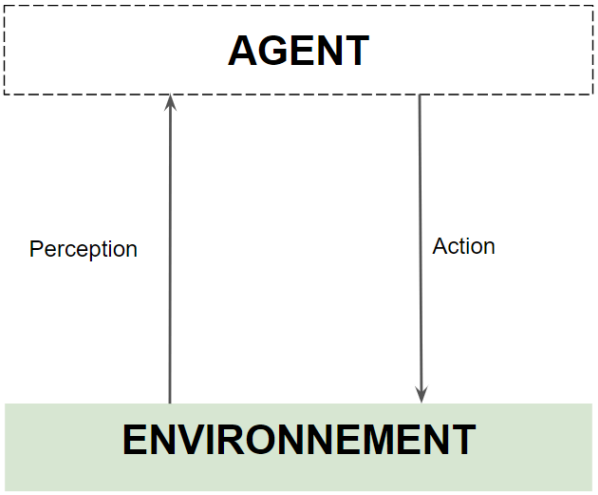


Figure 4: L'interaction perception-action de l'agent avec son environnement.

Il est également envisageable que l'agent dispose de plusieurs caractéristiques distinctes selon la thématique étudiée:

- un agent peut percevoir et collaborer avec d'autres agents [Jennings & Wooldridge, 1998];
- un agent peut apprendre de nouvelles compétences à travers un mécanisme d'apprentissage [Carmel & Markovitch, 1996];
- un agent peut être doté de capacités émotionnelles [Bourgais et al., 2016], sociales [Davidsson, 2002] ou avoir ses propres valeurs morales [Mercuur et al., 2019; Kammler et al, 2022].

1.2.2. La modélisation à base d'agents

Dans le processus de la modélisation à base d'agents, le modélisateur doit donc collaborer avec les thématiciens pour extraire de leurs discours les points de vue pertinents sur un sujet particulier. Les agents et leurs comportements sont ensuite initialisés dans un modèle, avant de lancer la simulation et de passer à l'interprétation des résultats par les thématiciens, le modélisateur, et potentiellement les informaticiens qui ont implémenté le modèle. Toutefois, ce processus pluridisciplinaire pose plusieurs contraintes :

- Le chercheur et l'audience ont des facultés cognitives limitées [Axelrod, 1997], ce qui pourrait encourager à produire des modèles simples selon le principe KISS *"keep it simple, stupid"* pour rendre le modèle compréhensible pour des non-informaticiens avec un petit nombre de paramètres, un environnement simple, peu d'entités, et un comportement très simple;
- D'un autre côté, il a été suggéré dans la communauté agent qu'il y a beaucoup à gagner à simuler des modèles plus riches, plus réalistes, mais aussi plus complexes [Drogoul et al., 2002].

Il est donc nécessaire d'adapter la complexité et l'intuitivité du modèle en fonction du sujet d'intérêt. Puisque notre sujet d'intérêt repose sur les normes et leurs influences dans le comportement des individus qui seront représentées par des agents, le présent travail s'insère dans les modèles dits de "simulation sociales" où l'on modélise des comportements humains et qui ont besoin d'un modèle plus descriptif et riche comme le défend [Adam & Gaudio, 2016, p.4-7]. Pour pouvoir obtenir des agents capables d'un processus décisionnel avec la descriptivité revendiquée pour ce type de simulation, il nous faut détailler :

i) comment est-ce qu'un agent engendre du comportement ?

ii) comment est-ce que les normes sont représentées et intégrées dans ce processus ?

iii) comment est-ce que les agents prennent en compte / violent les normes ?

iv) et enfin pour pouvoir juger de l'efficacité des normes: quels sont les impacts des normes sur les comportements des agents ?

Nous allons donc aborder successivement ces quatre questions, à commencer par décrire en section 1.3.Les architectures d'agent qui permettent aux agents d'engendrer des comportements proactifs et réactifs, et ce, de manière autonome.

1.3. Les architectures d'agent

Une architecture d'agent représente le processus décisionnel de l'agent basé sur un cycle de perception-action. Elle est définie par [Wooldridge & Jennings, 1995] comme étant le passage des spécifications de ce qu'est un agent (autonome, social, proactif, réactif) vers leur implémentation. Une architecture d'agent est l'architecture logicielle permettant aux agents de traiter des perceptions en entrée, et de produire des actions en sortie pour atteindre leurs objectifs : elle détaille les différents modules, leurs interactions et les algorithmes utilisés pour construire des agents intelligents [Maes, 1991].

Dans le cas d'un comportement proactif, le comportement en question sert à atteindre les objectifs de l'agent, tandis que dans le cas d'un comportement réactif il s'agit d'une simple réaction prédéfinie à un événement dans l'environnement de l'agent. Le rôle joué par

les architectures d'agent est explicité par la Figure 5 en laissant transparaître les mécanismes internes de l'agent.

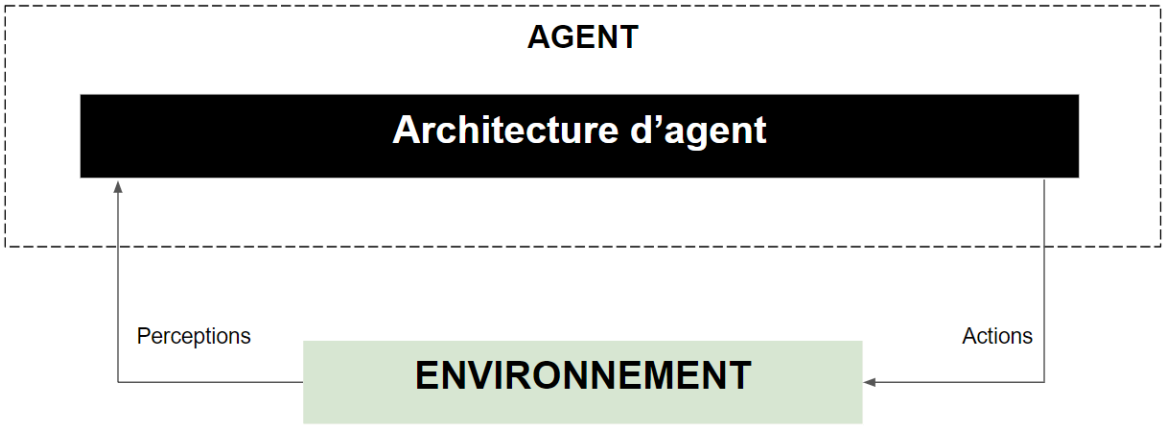


Figure 5: le rôle joué par l'architecture d'agent dans l'engendrement de comportement

Historiquement, nous dénombrons trois grandes familles d'architectures d'agents [Kleiner & Nebel, 2008], chronologiquement, nous avons :

- À partir de 1956 : émergence des architectures délibératives où les actions en sortie résultent d'un raisonnement symbolique;
- À partir de 1985 : émergence des architectures réactives basées sur du stimulus-réponse, où l'on aura une correspondance quasi-instantanée entre la perception et l'action;
- À partir de 1990 : émergence des architectures en couches ou hybrides qui combinent les architectures délibératives et les architectures réactives.

1.3.1. Les architectures Délibératives

Les architectures délibératives présentent un cycle de perception-action qui peut être, à un haut niveau d'abstraction, développé en un cycle PDA: perception - décision - action [Russell & Norvig, 1995], plus précisément, l'agent va :

- Percevoir le monde, et le représenter sous un format symbolique, y compris les connaissances sur l'agent lui-même mais aussi sur le monde;
- Décider grâce à un planificateur qui va délibérer entre plusieurs plans possibles;
- Choisir un plan et appliquer chacune des actions du plan (cf. Figure 6).

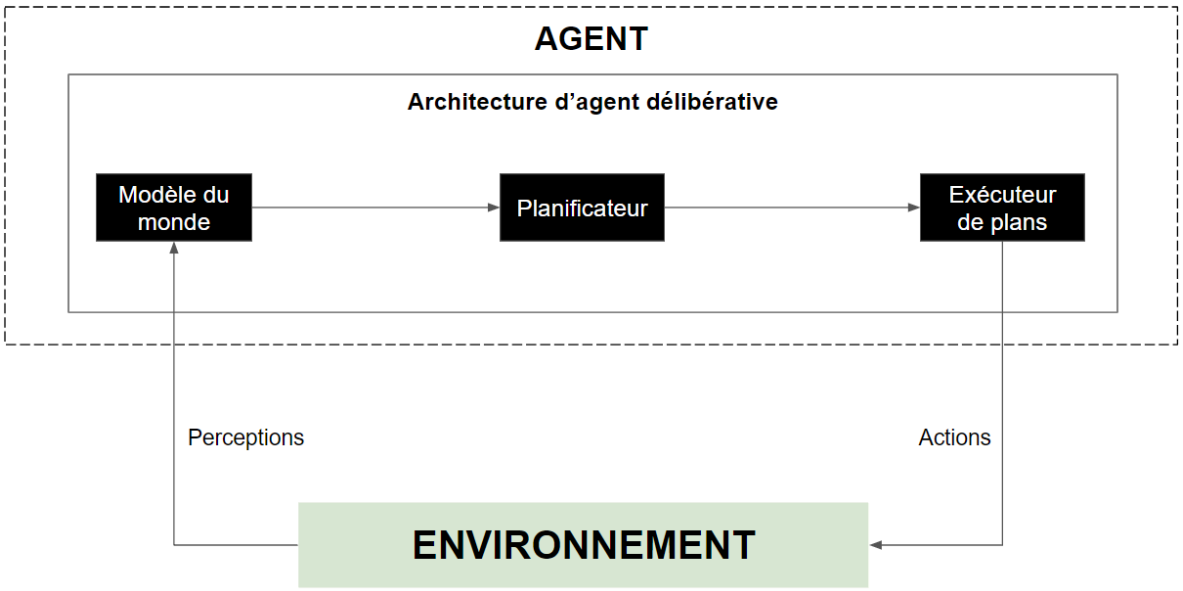


Figure 6: Schéma des composants d'une architecture délibérative [Ulieru, 2011]

1.3.1.1. Architecture BDI

L'architecture BDI ou Belief-Desire-Intention issue des travaux philosophiques de [Bratman, 1987] et été ensuite formalisée en tant qu'architecture d'agent par [Rao & Georgeff, 1991] qui est l'une des architectures les plus populaires. Il s'agit d'une architecture délibérative basée sur trois (03) concepts mentaux qui sont les croyances, les désirs, et les intentions représentés sur la Figure 7.

- Les croyances (*Beliefs*) qui sont construites à partir de la perception du monde;
- Les désirs (*Desires*) qui représentent les objectifs de l'agent et les manières envisagées pour les accomplir;
- Les intentions (*Intentions*) qui représentent la séquence d'actions ou plan que l'agent a décidé d'exécuter.

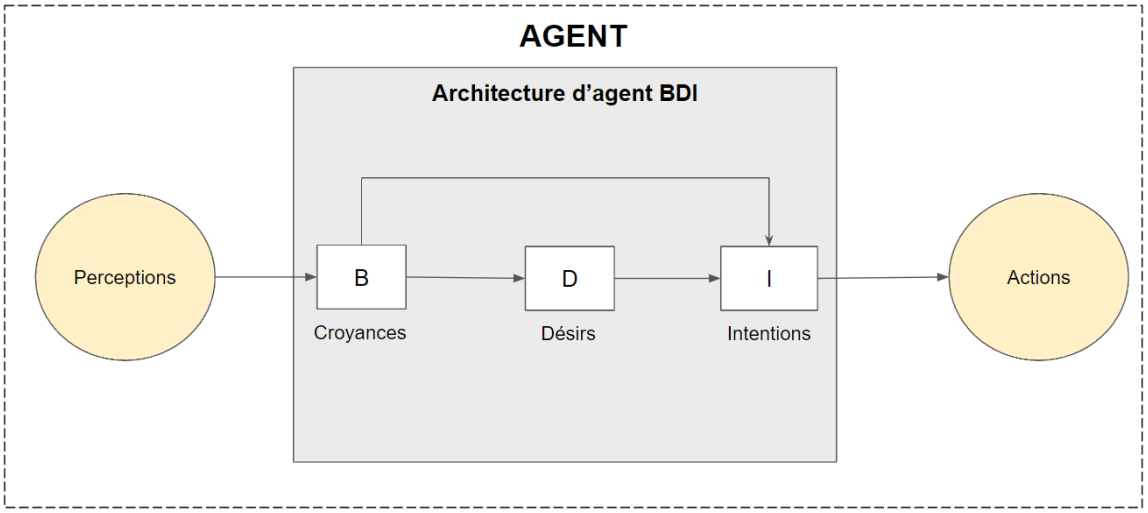


Figure 7: Structure globale des composants de l'architecture BDI

Nous pouvons expliciter ce schéma en un cycle délibératif PDA en listant l'ensemble des interactions entre les trois composants de base sur la Figure 8 : schéma détaillé de l'architecture BDI.

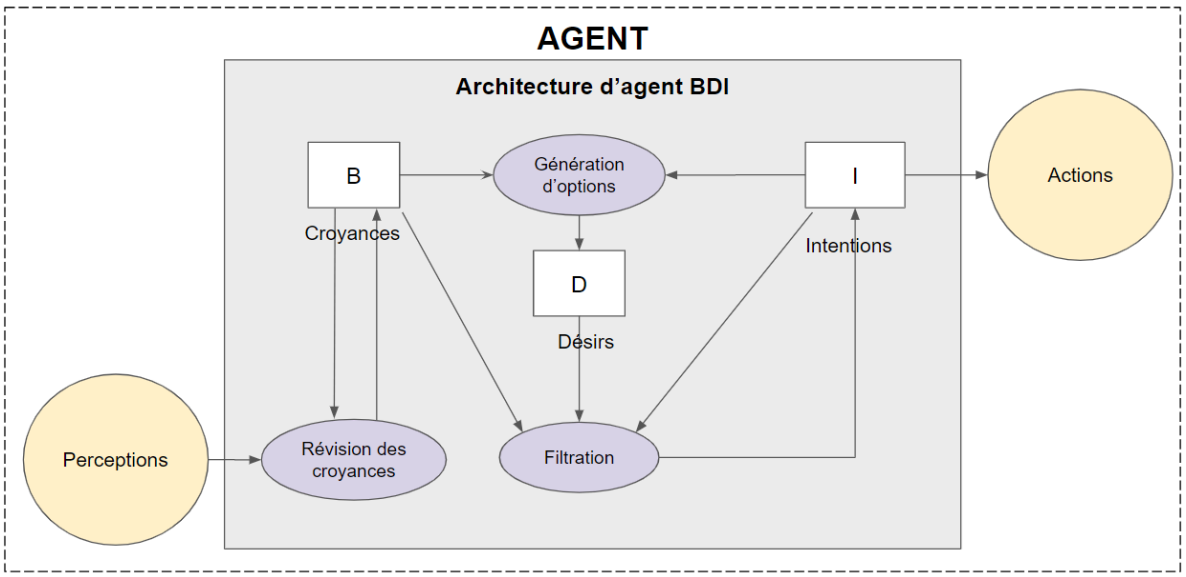


Figure 8 : schéma détaillé de l'architecture BDI

Son cycle de perception - décision - action (PDA) en tant qu'architecture délibérative peut se résumer aux étapes suivantes :

1. Pour percevoir, l'agent va mettre à jour sa base de croyances sur le monde à l'aide d'une fonction de révision de croyance appelée la *Belief Revision Function* (Brf);
2. Pour décider, elle génère un ensemble d'options disponibles (ses désirs, ainsi que les plans choisis pour y parvenir) en tenant compte de ses intentions et croyances actuelles; l'agent doit ensuite les filtrer pour générer de nouvelles intentions, en partant toujours de ses croyances, désirs, et intentions existantes;
3. Pour agir, les intentions de l'agent, i.e. l'ensemble des action(s) ou des plans qui permettront de les réaliser, sont exécutées.

Pendant longtemps, les langages de programmation basé sur l'architecture BDI tel que PRS [Georgeff & Ingrand 1989], IRMA [Bratman et al, 1987], dMARS [Rao & Georgeff, 1995] ont opté pour des plans pré-compilés dans une bibliothèque de plans pour réduire le temps de calcul. Plusieurs travaux s'intéressent également à des algorithmes plus performants pour guider la sélection de plans prédéfinis [Sardina et al, 2006].

Avec l'avancée des technologies en termes de création de plans de zéro tels que Graphlan ou RePop [Blum & Faust, 1997; Nguyen & Kambhampati, 2001], des langages de programmation BDI permettent la construction de nouveaux plans à partir de zéro pour réaliser une intention, e.g. [Meneguzzi et al., 2004; Walczak et al., 2007]. Cette faculté permet d'augmenter l'autonomie des agents BDI, en exploitant la notion d'objectif : lorsqu'il n'y a pas de plans prédéfinis qui atteignent un objectif, l'agent peut concevoir un plan pour y arriver, ou bien trouver un moyen pour rendre applicable un plan qui permet de l'accomplir [de Silva et al., 2009].

Vu ce cycle PDA, l'architecture BDI se distingue par un très haut niveau d'abstraction, ce qui lui vaut l'avantage de pouvoir représenter de manière simple un raisonnement complexe pour les thématiciens. Le comportement de l'agent est donc non seulement conditionné par les intentions, mais également par les désirs, les plans, et les croyances de l'agent lorsqu'ils interviennent dans l'élaboration des plans.

1.3.1.2. Architectures cognitives

Les architectures cognitives sont des architectures d'agent délibératives qui se basent sur une théorie cognitive spécifique. À ce jour, hormis les travaux philosophiques BDI de Bratman, il n'y a pas vraiment de théorie standard adoptée par les SMA, et chaque architecture cognitive se base plutôt sur sa propre théorie : par exemple, l'architecture CLARION [Sun et al., 2001] est basée sur les travaux de [Sun, 1994], l'architecture SOAR est basée sur les travaux de [Newell, 1994]; et ACT-R [Lebiere & Anderson, 1993] est basée sur les travaux de [Anderson, 1983].

Cette hypothèse sur une pléthore de théories sous-jacente aux architectures cognitives est mis en évidence dans la revue de [Ye et al, 2018] avec une analyse d'une cinquantaine d'architectures cognitives au cours des 20 dernières années avec une taxonomie détaillant si l'architecture prend en compte : les perceptions, les actions, la mémoire, le raisonnement, l'apprentissage, l'attention, les émotions, la planification, les interactions, la motivation. D'autres revues extensives des architectures cognitives sont proposées dans [Chong et al., 2007] et [Stollberg & Rhomberg, 2006]

Dans le cadre des simulations sociales qui intéressent le présent travail, nous allons seulement présenter quelques-unes de ces architectures cognitives qui pourraient offrir différents points de vue pertinents sur les simulations sociales, et les comparer au cycle PDA de l'architecture BDI à savoir : SOAR et CLARION.

Soar. se base sur la théorie de [Laird, 1986; Laird et al., 1987; Laird et al., 1990]. Ses implémentations sont centrées autour d'un dérivé de la recherche dans un espace d'états, à l'aide d'opérateurs pour atteindre les objectifs fixés en ajoutant entre autres la notion *d'impasses,* et un système d'apprentissage. Les opérateurs correspondent à des actions, munis de préconditions et de post-conditions. Le cycle PDA de Soar correspond respectivement à 4 phases :

1. la *INPUT PHASE* qui va communiquer tout changement sur l'environnement et les stocker dans la partition dédiée aux perceptions (appelée *input-link*) dans la mémoire;
2. la *ELABORATION PHASE* qui liste toutes les stimulus-réponse (dites des *règles de production* dans SOAR) dont la précondition est vérifiée dans l'état actuel (y compris

les impasses du cycle précédent), ceci permet de mettre à jour le modèle du monde, puis propose l'ensemble des opérateurs applicables;

3. la *DECISION PHASE* qui va trier les opérateurs à disposition en fonction des préférences. Si un seul opérateur subsiste, Soar passe à la phase suivante, sinon, il génère une impasse pour décider entre les opérateurs équivalents au prochain cycle;
4. la *APPLICATION PHASE* qui va appliquer l'opérateur et appliquer des changements sur l'environnement pour être de nouveau perçus par la *INPUT PHASE*.

Une correspondance entre ces phases et le cycle PDA de l'architecture BDI est présentée dans [Wray & Jones, 2005] et est reprise dans le Tableau 2.

Tableau 2: Correspondances entre l'architecture SOAR et BDI

BDI	**SOAR**
Perception	INPUT PHASE
Brf (Révision des croyances)	ELABORATION PHASE (liste des règles de productions vérifiées)
Générations d'options (Désirs)	ELABORATION PHASE (proposition d'opérateurs applicables)
Filtration des désirs (en Intentions)	DECISION PHASE
Choix et exécution d'un plan	APPLICATION PHASE

D'un point de vue pratique, selon [Sun, 2007], pour pouvoir utiliser Soar, il faut une large connaissance de ce qu'est un état et un opérateur doit être acquis avant de pouvoir l'utiliser.

CLARION. en comparaison, est une architecture cognitive basée sur les travaux cognitifs de [Sun et al., 2001] ,dont les composants présentent tous deux niveaux : 1) un niveau inférieur avec un algorithme d'apprentissage par rétropropagation du gradient pour obtenir des connaissances implicites, 2) un niveau supérieur avec des connaissances explicites sous une forme symbolique, tirées des connaissances implicites. CLARION dispose de sous-systèmes pour contrôler les actions, les motivations, les perceptions, et le fonctionnement général de l'architecture. Sommairement, le processus PDA de CLARION est de :

1. percevoir l'état actuel *x* du monde;
2. calculer au niveau inférieur l'ensemble des *Q-valeurs* (une évaluation de la qualité) associées aux actions possibles dans *x*;
3. calculer au niveau supérieur l'ensemble des *Q-valeurs* associées aux règles applicables dans *x*;
4. Comparer ces deux ensembles de valeurs, et choisir une action appropriée *a*;
5. Exécuter l'action *a* pour obtenir l'état suivant *y* et le renforcement *r*;
6. Mettre à jour le niveau inférieur par *Q-learning*;
7. Réviser l'ensemble des règles du niveau supérieur;
8. Revenir à l'étape 1.

Dans [Naveh & Sun, 2006], des extensions possibles de CLARION pour intégrer les normes sont proposées à l'aide de deux structures organisationnelles : a) une équipe, où les décisions sont fournies par un ensemble d'agents, b) une hiérarchie, où les décisions sont fournies par un agent supérieur, où les informations au sein d'une organisation sont : a) distribuées: chaque agent aura des informations différentes, ou b) bloquées: où les agents auront accès aux mêmes informations. L'intégration des normes dans le sous-système est prise en charge par un nouveau sous-système appelé la *Motivation Subsystem*.

À travers ces deux exemples nous pouvons notamment voir que les détails des architectures d'agent cognitives peuvent différer en fonction des travaux cognitifs associés, en intégrant plusieurs types de mémoires, ainsi que différents composants pour représenter les motivations des agents, la gestion des actions, et la mise en place des plans. Si en intelligence artificielle, Soar est largement reconnu, dans le domaine des SMA, l'architecture BDI reste la théorie de facto pour concevoir des agents délibératifs.

1.3.1.3. Architecture d'agent à réseau de neurones

Plutôt que d'avoir un ensemble de plans prédéfinis comme avec les agents BDI classiques, les architectures d'agent à réseau de neurones proposent d'engendrer du comportement par un réseau de neurones entraîné typiquement à l'aide de soit 1) un algorithme génétique, ou 2) un algorithme de back-propagation. Les agents à réseaux de neurones sont des cas particuliers des agents réactifs, où le stimulus-réponse se base sur un

réseau de neurones forgé par un apprentissage supervisé sur un ensemble de données. Elles sont notamment utilisées :

- dans la modélisation du comportement optimal des micro-organismes en quête de nourriture dans un environnement clos [Lahodiuk, 2013]
- dans les travaux de [Dembytskyi & Dorogyy, 2017] pour modéliser des comportements humains durant un incendie.

L'algorithme génétique [Holland, 1975] est une méthode de recherche stochastique visant à minimiser ou maximiser une fonction heuristique, dont le principe est de partir d'une population initiale de gènes (des variables), qui seront assemblés deux à deux en chromosomes pour former des solutions potentielles au problème, et qui seront évaluées. Les meilleurs chromosomes seront ensuite utilisés pour former une nouvelle population, qui aussi seront assemblées à leur tour, jusqu'à ce que le critère de terminaison soit atteint, par exemple : 1) la recherche s'arrête après un nombre fixe d'itérations, et on prend le meilleur chromosome obtenu, ou 2) la recherche s'arrête lorsque le meilleur chromosome ne voit pas sa fonction heuristique croître pour un nombre défini d'itérations, celle-ci est alors choisie.

Ce type d'algorithme permet d'obtenir des comportements d'agents qui produisent du comportement de plus en plus efficace par apprentissage. Toutefois, ces agents présentent: 1) un temps d'apprentissage nécessaire pour les agents pour aboutir à un comportement efficace, et 2) une difficulté d'explication du comportement aux thématiciens, puisque l'agent aura produit son comportement en se basant sur le croisement aléatoire des gènes de la population initiale, ce qui rend difficile d'expliquer "pourquoi" l'agent a pris tel décision, plutôt qu'une autre sans introduire une théorie sur l'algorithme génétique.

Quant à l'algorithme de back-propagation (BP) comme son nom l'indique, il s'agit d'une propagation de l'erreur comparée au comportement attendu. Cette erreur représente la distance entre le comportement attendu et le comportement produit actuellement, qui est propagée dans tous les neurones du réseau. Le poids de chaque réseau de neurones est ensuite mis à jour jusqu'à ce que l'erreur soit relativement petite et que les poids convergent [Chong et al., 2011].

1.3.1.4. Architecture d'agent en temps réel

Une des innovations de plus en plus proposées en termes d'architecture BDI, est la planification en temps réel (RT-BDI ou *Real Time BDI*). Si les agents BDI, et plus généralement les architectures délibératives peuvent raisonner de manière autonome sur des problèmes complexes, il se peut que lorsque le plan s'exécute, il ne soit plus pertinent par rapport à la situation actuelle vu le temps de calcul. Pour pouvoir produire des plans qui respectent les échéances, les auteurs tels que [Traldi et al., 2022; Alzetta et al., 2020] fournissent une représentation formelle des échéances, non seulement sur l'échéance pour trouver / accomplir les plans, mais aussi l'échéance sur la validité des objectifs. Par exemple, pour [Traldi et al., 2022] les éléments qui constituent un désir sont :

- Une précondition pour être activée;
- Un objectif décrivant la condition qui doit être achevée;
- Une date limite qui représente un instant où l'on souhaite réaliser le désir;
- Une priorité sous la forme d'un entier représentant la priorité du désir pour l'agent.

Quant aux intentions, elles sont composées de :

- Soit un plan atomique qui comporte une action avec i) son instant de début, ii) sa durée, iii) la condition qu'il faut maintenir durant toute la durée du plan, iv) son effet sur les croyances à terme;
- Soit d'un plan séquentiel qui oblige l'exécution d'un ensemble de sous-plans;
- Soit d'un plan parallèle qui spécifie l'exécution potentiellement en parallèle d'un ensemble de sous-plans.

L'idée principale est de rajouter à l'architecture en temps réel une couche qui sera chargée du suivi de l'exécution des plans. Cette architecture se focalise donc sur la cohérence temporelle des plans par rapport aux échéances des objectifs de l'agent.

Une de leurs limites est qu'elles ne prennent pas en compte le temps qu'il faut pour exécuter le cycle de perception-action, plus précisément pour trouver un plan. Pour pallier cela, les architectures en temps réel comme RT-BDI [Traldi et al., 2022] doivent se baser sur des plans manuellement définis, et le plan choisi est le plan qui non seulement accomplit l'objectif, mais qui l'accomplit en respectant son échéance. La création de nouveaux plans

d'action par un planificateur n'est invoquée que quand les plans prédéfinis ne permettent pas d'accomplir l'objectif dans les temps.

1.3.2. Les Architectures Réactives

Une architecture réactive est une architecture qui se base sur des correspondances de perception-action prédéfinies, il n'y a point de délibération dans ce type d'architecture. Comparées aux architectures délibératives, les architectures réactives ont besoin de peu de ressources, et peuvent réagir plus vite, bien qu'elles ne soient pas aussi flexibles, et ne permettent pas de représenter des comportements proactifs directement (cf. Figure 9).

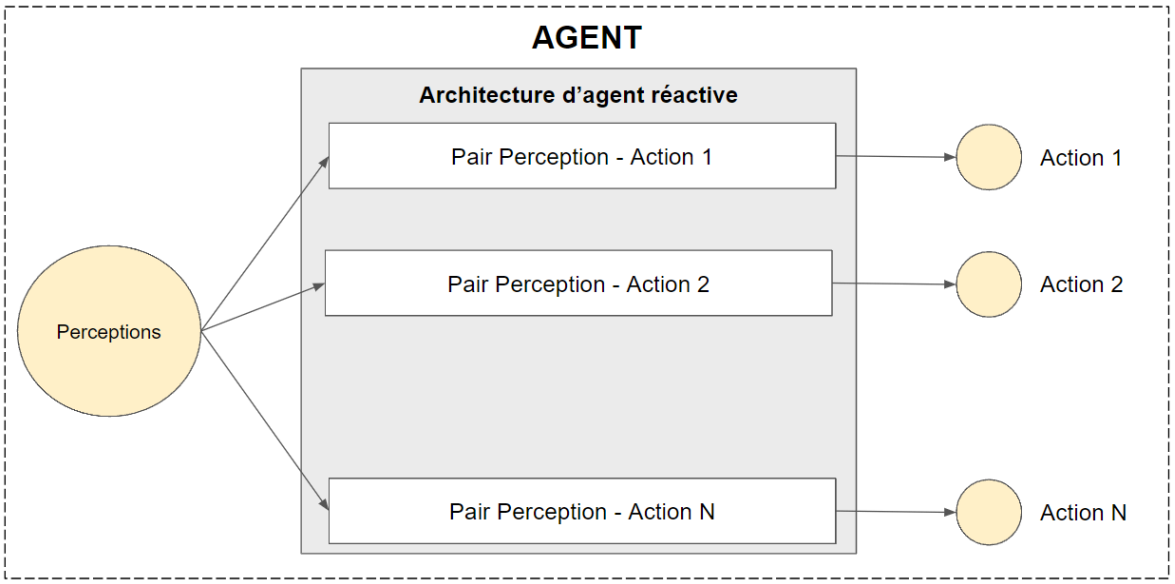


Figure 9: Représentation de l'architecture réactive

Quelques exemples de travaux ayant utilisés une architecture réactive seraient :

- les travaux de [Helbing et al, 2000] où les comportements des agents sont représentés par des forces calculées sous la forme d'équations électromagnétiques;
- l'architecture de subsomption de [Brooks, 1987] où les comportements réactifs sont disposés en couches où les couches supérieures peuvent influencer les couches inférieures. Sa thèse montre ainsi que des comportements complexes peuvent être obtenus à l'aide de l'interaction de plusieurs comportements réactifs simples.

Les architectures réactives présentent deux avantages notables : 1) la simplicité, car les comportements s'articulent sous la forme d'une condition simple : "si telle perception, alors faire telle action", 2) la coordination des comportements délibératifs par inhibition, car quand plusieurs réactions sont possibles, le système choisit une règle de perception-action en inhibant d'autres, notamment dans l'architecture de subsomption de [Brooks, 1987], ce qui leur permet d'obtenir des comportements complexes comme les architectures délibératives.

1.3.3. Les Architectures Hybrides

` Les architectures hybrides combinent les avantages des architectures délibératives BDI et les architectures réactives en les disposant en couches verticales ou horizontales [Chin et al., 2014]. Ceci permet d'avoir d'une part la réactivité des architectures réactives, mais aussi la proactivité des architectures délibératives. Il est également possible d'avoir diverses combinaisons d'architectures en couches également, par exemple en ayant plusieurs couches réactives, plusieurs couches délibératives, et une couche pour l'apprentissage. La disposition des couches peut se faire de manière verticale ou horizontale :

a) Dans une architecture horizontale (Figure 10), chaque couche détermine les actions qu'il faut effectuer et une fonction médiatrice sera nécessaire pour déterminer quelle couche primera devant les autres;
b) Dans une architecture verticale (Figure 11), la perception passe par une couche de base, et peut transiter vers des couches supérieures si la couche actuelle ne parvient pas à trouver les actions à effectuer, ou si l'on souhaite trouver un meilleur comportement.

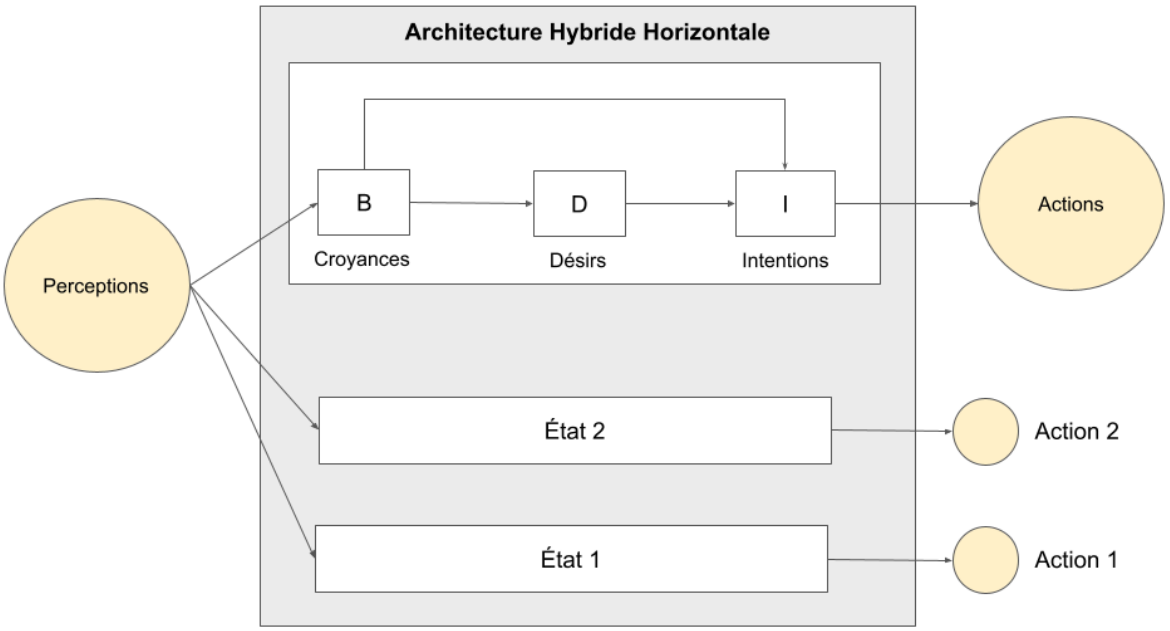


Figure 10 : Illustration d'une architecture d'agent hybride horizontale

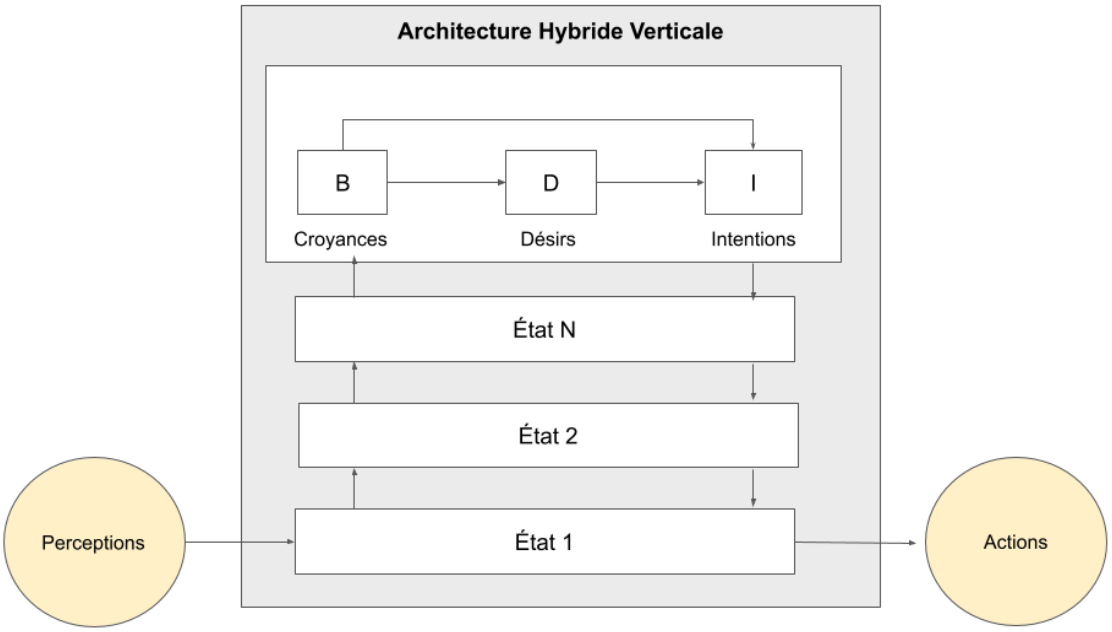


Figure 11 : Illustration de l'architecture hybride verticale

Quelques exemples d'architectures hybrides connues sont ACT-R [Anderson & Lebiere, 2003], CLARION [Sun & Alexandre, 1997], DUAL [Kokinov, 1994] et LIDA [Franklin & Patterson, 2006].

1.3.4. Synthèse sur les architectures d'agents

Nous avons distingué à la suite de cette revue sur les architectures d'agent trois grandes familles d'architectures :

- Les architectures délibératives : basées sur des théories philosophiques ou cognitives, coûteuses en termes de calcul, mais très utiles pour engendrer des comportements adaptés à des situations très complexes, telles que les simulations sociales avec des êtres humains;
- Les architectures réactives : très performantes en termes de calcul, mais peu flexibles quant aux comportements qu'elles peuvent engendrer ;
- Les architectures hybrides : capables à la fois de comportements réactifs performants, et de comportements complexes, souvent utilisés dans plusieurs implémentations, telles que SOAR ou CLARION.

Au terme de cette section, la question principale est de savoir : *"Quelle architecture est la plus appropriée pour prendre en compte les normes?"*. Le choix d'une architecture par rapport à une autre dans la cadre d'une modélisation à base d'agent dépendra :

1) de la complexité des comportements que l'on souhaite modéliser : une architecture plus simple telles que les architectures réactives conviendrait pour des comportements sans réelle capacité de réflexion : les arbres, les lacs, ou les feux; tandis que pour modéliser des comportements plus complexes motivés par un objectif, voire des normes, il est préférable d'opter pour des architectures délibératives [Adam et al., 2011];
2) du niveau d'abstraction et de la facilité de compréhension du modèle pour des non-informaticiens: une architecture plus compréhensible favorise la confiance des thématiciens et des modélisateurs, et facilite le processus de modélisation ainsi que les discussions autour du modèle [Adam et al., 2011];

3) de la granularité de l'objet d'observation: si l'accent est mis sur l'observation du comportement global du SMA avec les normes, alors une architecture simple peut suffire; par contre, si l'accent est mis sur l'observation des comportements individuels des agents face aux normes, une architecture plus fine et plus riche est préférable [Carley et al., 1998];
4) de l'échelle du modèle : bon nombre d'architectures plus réalistes consomment déjà beaucoup de temps et de ressources pour calculer le comportement d'un seul agent; de ce fait, en faire tout un SMA à grande échelle, avec des milliers d'agents semble peu pratique. Ces problèmes de passage à l'échelle sont confirmées dans les travaux de [Hindriks et al., 2010] où les architectures plus complexes rencontrent des problèmes dès lors que le nombre d'agents croît;
5) de la nature du discours des thématiciens à disposition : la qualité des données fournies par les discours des thématiciens peut grandement affecter le choix de l'architecture d'agent; si le modélisateur ne dispose que des informations objectives globales (nombre de ménages, les revenus, etc.), il lui est très difficile de décrire un comportement individuel plus riche à l'aide d'une architecture telle que l'architecture BDI [Gil-Quijano et al, 2007]. Il reste cependant possible de capter et retranscrire des données cognitives sous la forme de composants mentaux à l'aide d'une modélisation participative [Bousquet & Trébuil, 2005] et des jeux sérieux [Wein & Labiosa, 2013];
6) de la multiplicité des théories sous-jacentes à l'architecture : si l'architecture utilisée a sa propre théorie sous-jacente (par exemple les architectures cognitives telles que ACT-R ou Soar qui se basent toutes sur différentes théories), il serait difficile de réutiliser le modèle et de collaborer avec d'autres personnes. La présence d'une théorie unifiée et constante, permet de faciliter l'interprétation du modèle, sa réutilisation éventuelle par d'autres collaborateurs, et ce faisant, il permet d'étendre la pertinence du modèle par rapport au domaine d'application;

Si ces architectures d'agent étalent les différents composants mentaux et leurs interactions, la manière dont elles sont représentées formellement et comment l'agent trouve un plan peut toujours s'implémenter de différentes manières. Toutefois, il reste à détailler comment ces architectures prennent en compte les normes : si les architectures d'agent restent des indications théoriques sur comment produire du comportement d'agent, il nous faut explorer comment les normes sont prises en compte dans ces architectures.

1.4. Les architectures d'agent normatives

Pour reproduire un comportement humain réaliste, l'architecture d'agent doit non seulement produire un comportement en fonction de ce qu'il veut faire, mais aussi en fonction de ce qu'il doit ou ne doit pas faire, i.e. il doit intégrer les normes. Dans cette optique, les architectures d'agent classiques sont étendues pour produire des comportements adéquats par rapport aux normes qui nous intéressent [Luck et al., 2013].

Ces architectures dites *Normas* (ou *Normative Multiagent Systems*) représentent tout simplement un ensemble d'agents gouvernés par les normes qui leur prescrivent un comportement idéal. Selon [Boella et al., 2006]. Les agents dans les SMA normatifs disposent des mécanismes de représentation, de communication, de distribution, de détection, de création, de modification, et d'application des normes, y compris leur violation [Boella et al, 2007].

Les SMA sont des systèmes ouverts, ce qui implique l'interaction d'agents hétérogènes[4] et autonomes qui doivent se conformer à plusieurs normes pour atteindre une stabilité souhaitée, et qui peuvent être implémentées de différentes façons, comme le cas des socio-écosystèmes traités dans [Rakotonirainy et al., 2015a]. Dans le cadre des institutions électroniques, les normes sont considérées comme des moyens de coordination pour ces agents, en s'assurant qu'il n'y a jamais de violation des normes, ce qui restreint sévèrement l'autonomie de l'agent en lui imposant des actions dès que les normes sont applicables, comme l'exemple des travaux sur AMELI [Esteva et al., 2004; Dignum & Dignum, 2001]. Cependant, en tant qu'entités autonomes, les agents devraient pouvoir décider de violer ou de respecter une norme, pour recevoir une certaine récompense ou pour éviter une certaine sanction, par exemple.

La prochaine section détaillera où cette délibération sur les normes se situe, dans le cycle de vie des normes dans les architectures d'agents de leur perception à leur application ou violation.

1.4.1. Cycle de vie des normes dans les architectures d'agent

Le cycle de vie des normes passe par quatre activités selon les modèles existants dans la littérature [Savarimuthu, 2011; Hollander & Wu, 2011]. Nous dénotons parmi les étapes du

[4] Des agents hétérogènes sous-entend ici que les agents développés avec différents langages, par différentes personnes, avec différents objectifs

cycle de vie d'une norme : i) la perception des normes : l'agent identifie les normes applicables, ii) l'adoption des normes : l'agent reconnaît qu'une norme lui est applicable, iii) la délibération des normes : l'agent évalue les influences de la norme sur ses propres objectifs et décide de le respecter ou non à l'aide de différentes stratégies, iv) l'exécution d'actions relatives aux normes : une fois la norme adoptée, les objectifs de l'agent et l'environnement sont mis à jour et le cycle reprend [Mahmoud et al., 2014]. Ce cycle de vie des normes dans le comportement des agents est représenté sur la Figure 12.

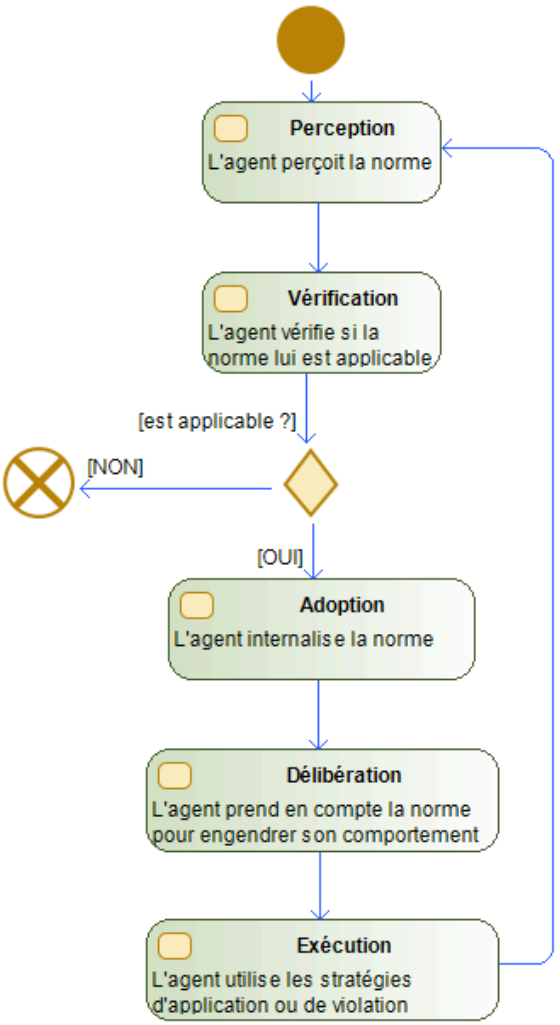


Figure 12: Cycle de vie des normes dans les architectures d'agent normatives, adapté de [Mahmoud et al., 2014]

Les impacts des normes sur le comportement des agents qui nous intéressent se situent dans le processus de délibération : "Comment cette norme adoptée va impacter le comportement de l'agent? Par quelle(s) stratégie(s) ?" notamment si l'agent n'a pas de plans

prédéfinis pour y faire face. Pour comprendre comment les architectures d'agent classiques ont été modifiées pour prendre en compte les normes, nous répondrons dans les prochaines sections à deux questions principales :

1) Comment les normes sont prises en compte dans les architectures réactives ?
2) Comment les normes sont prises en compte dans les architectures délibératives ?

Parmi les architectures normatives dans la littérature, nous pouvons citer entre autres :

- [Oren et al., 2011] qui adopte une architecture BDI basée sur une fonction d'utilité où, dans un ensemble de plans déterminés, on prend en compte la baisse d'utilité d'une violation d'une interdiction ou d'une obligation, et le gain d'utilité d'un respect d'une norme. Pour les cas des permissions, elles sont associées à des pertes en utilité lorsqu'elles sont utilisées (puisqu'utiliser une permission inspire un sentiment de méfiance). Ils se reposent essentiellement sur une bibliothèque d'action, mais en prenant en compte l'utilité en violant ou en respectant les normes pour décider quelle norme respecter ou violer, et ainsi obtenir un plan optimal;
- [Kollingbaum & Norman, 2003] à travers leur architecture NoA explorent les architectures réactives pour déterminer comment les agents décident de prendre en compte les normes, notamment dans le cas où un agent décide d'adopter une obligation pour accomplir une condition *p* alors qu'il lui est déjà interdit d'accomplir *p*, ou bien que tous les plans pour accomplir *p* violent tous une autre interdiction que l'agent a déjà prise en compte.

1.4.2. Architectures réactives normatives

1.4.2.1. NoA

Dans les travaux de [Kollingbaum & Norman, 2003] sur leur architecture d'agent réactive et normative nommée NoA qui filtre les actions ou désirs en conflit avec les normes. Le comportement des agents est conditionné par trois éléments :

a) Un ensemble de croyances;

b) Un ensemble de plans prédéfinis composé à l'instar de l'architecture BDI de préconditions pour les activer, et de leurs effets une fois accomplis, et enfin le corps à proprement parler du plan;
c) Un ensemble de normes (obligations, interdictions, permissions) définies par une condition d'applicabilité et d'expiration.

Les obligations motivent les agents à choisir certains plans pour les accomplir. Les interdictions restreignent les plans qui peuvent être choisis. Les permissions permettent de passer outre les interdictions. Se basant sur la logique des responsabilités sur les états et les actions dans [Norman & Reed, 2000], l'architecture NoA adopte une distinction claire entre la responsabilité pour accomplir un certain état du monde et une action. Puisque les normes vont venir influencer le choix des plans prédéfinis, les auteurs définissent formellement différents degrés de conflits et de cohérence entre les normes. Ces conflits permettent à l'agent de décider s'il va prendre en compte ou violer des obligations et des interdictions en fonction de ces conflits.

Pour une obligation d'accomplir un état *s*: il est dit "fortement incohérent" avec un ensemble d'états interdits S_F et un ensemble d'états obligatoires S_O si et seulement si :

$$\forall p \in PLANS. \quad Si \quad s \in effects(p)$$

$$Alors \quad S_F \cap effects(p) \neq \emptyset$$

$$ou \quad S_o \cap neg.effects(p) \neq \emptyset$$

Où $effects(p)$ désigne les effets du plan p, et $neg.effects(p)$ désigne la négation de $effects(p)$ et PLANS désigne l'ensemble des plans pour accomplir l'obligation. Ainsi, pour toute nouvelle obligation visant à accomplir un état *s*, un conflit émerge si celle-ci ne contient pas pour effet un état interdit, et qu'elle ne contient pas la négation d'états obligatoires existants.

De cette définition d'une forte incohérence, sont déduites deux autres relations : la forte cohérence et la faible cohérence. Une nouvelle obligation est fortement cohérente avec les normes existantes s'il n'y a nécessairement aucun conflit possible, c'est-à-dire, une obligation est en forte cohérence avec les normes existantes si et seulement si :

$$\forall p \in PLANS. \quad Si \quad s \in effects(p)$$

$$Alors \quad S_F \cap effects(p) = \emptyset$$

$$ou \quad S_o \cap neg.effects(p) = \emptyset$$

La dernière forme de relation d'une nouvelle obligation est la faible cohérence, où il existe au moins un plan parmi l'ensemble des plans possibles qui permet d'accomplir *s* sans enfreindre les états interdits et les états obligatoires actuels, c'est-à-dire, elle est en faible cohérence si et seulement si :

$$\forall p \in PLANS\ tel\ que. \quad s \in effects(p)$$

$$et \quad S_F \cap effects(p) = \emptyset$$

$$et \quad S_o \cap neg.effects(p) = \emptyset$$

Quant aux interdictions, elles restreignent les plans que l'agent peut choisir, en impactant sur la cohérence de l'ensemble des normes, qui peuvent rester telles quelles, ou bien qui peuvent devenir un ensemble de normes en forte ou faible cohérence ou en forte incohérence. Les permissions permettent quant à elles d'améliorer un ensemble de normes, en les rendant soit en forte cohérence, soit en faible cohérence.

En résumé, les travaux de [Kollingbaum & Norman, 2003] décrivent comment les agents réactifs choisissent d'adopter ou non une norme, et contraignent le choix d'un plan parmi un ensemble de plans prédéfinis, puisque l'agent NoA cherchera à maximiser la cohérence de son ensemble de normes. Chaque norme est ainsi traitée une par une pour analyser l'apport d'une norme en termes de cohérence : il s'agit d'un mécanisme de filtration. Cette analyse de cohérence se base sur l'analyse des effets de plans permettant d'accomplir une certaine obligation, qui peut être dégradée par des interdictions, et améliorée par des permissions.

1.4.2.2. De Campos

Dans l'objectif d'étendre les agents réactifs (dits "*reflex agents*") avec des concepts pouvant prendre en compte les normes, [de Campos et al., 2012] propose de prendre en compte 3 modalités déontiques (les obligations, les interdictions et les permissions), et qui peuvent se porter sur soit : a) des actions, ou b) des états du monde. Une norme est représentée de manière classique avec :

1) Une modalité déontique qui peut être une interdiction, une obligation. Les permissions ne sont pas explicitement considérées mais, toute action non interdite, et non obligatoire est une action permise;
2) La condition d'activation de la norme;
3) La condition d'expiration de la norme;
4) Les attributs de la norme;
5) Les objectifs de la norme qui peuvent soit être des actions, soit des états du monde;
6) Les sanctions et les récompenses pour avoir violé ou respecté une norme sous la forme d'un entier.

Pour prendre en compte les normes dans les architectures réactives, [de Campos et al., 2012] propose de transformer les normes en tant qu'extension des règles perception-action. Pour cela, ils introduisent aux agents réactifs trois groupes de perception-action, chacun rattaché à un opérateur déontique et des sanctions en cas de violation :

- Le groupe des obligations : si un événement correspond aux règles de perception-action de ce groupe, il sera effectué par l'agent;
- Le groupe des interdictions : si un événement correspond aux règles de perception-action de ce groupe, il ne sera pas exécuté par l'agent;
- Le groupe des permissions : si un événement correspond aux règles de perception-action de ce groupe, il peut être exécuté ou non exécuté par l'agent.

Leur algorithme de prise en compte des normes considère que :

- Si une action est obligatoire, alors l'agent doit l'exécuter seulement si elle n'est pas interdite par une autre norme;
- Si une action est interdite, alors l'agent doit chercher une autre action qui est permise.

- Si une action n’est ni obligatoire ni interdite, l'agent peut choisir l’action qui est alors considérée comme permise.

Ainsi, le cycle perception-action de l’architecture réactive normative se décompose en 4 étapes pour [de Campos et al., 2012] en réponse à un stimulus:

1) Il recherche d'abord les règles dans le groupe des règles obligatoires afin de trouver les actions qui doivent être effectuées en réponse aux stimulis et qui ne sont pas interdites;
2) S’il existe une action interdite parmi les actions obligatoires à exécuter, la fonction inhibe les règles du groupe des règles obligatoires, et recherche les règles du groupe des règles interdites pour trouver les actions qui ne sont pas interdites et qui peuvent être exécutées ;
3) s'il n'y a pas d'interdiction, la fonction sélectionne l'action qui doit être exécutée à la lumière des obligations;
4) et enfin, dans le cas où il n'existe ni obligation ni interdiction, la fonction recherche des règles dans le groupe des règles de permission afin de trouver une action qui peut être exécutée dans l’état du monde actuel.

Globalement, la stratégie des agents normatifs et réactifs est simple : maximiser les récompenses et minimiser les sanctions. Pour les obligations dans [de Campos et al., 2012], le principe est de 1) choisir une action obligatoire et non-interdite, sinon, 2) choisir une non-interdite mais qui est permise dont les préconditions sont satisfaites dans l’état actuel du monde. Pour les interdictions, les règles dans le groupe des Permissions qui incluent l’action interdite sont inhibées, et on exécute une autre action permise à la place.

Cette extension se base donc sur un mécanisme de filtration qui ajoute de nouvelles règles de perception-action, et qui choisit parmi les actions à effectuer en fonction de la quantité de sanctions / récompenses obtenues. Cette hiérarchie entre les obligations, les interdictions et les permissions reste toutefois limitée puisque : 1) il faut quantifier les récompenses et les sanctions de chaque norme; 2) les interdictions sont absolues, elles empêchent les actions obligatoires de s’exécuter, et ne peuvent être résolues ou contournées.

1.4.3. Architectures délibératives normatives

Dans la grande majorité des architectures délibératives normatives, on observe deux manières générales de prise en compte des normes, soit a) par filtration : supprimer toute source potentielle de violations de normes au niveau des plans et des objectifs [Meneguzzi & Luck, 2009], soit b) par génération d'objectifs normatifs [Conte et al, 1999, Dignum, 1999].

La séparation des objectifs personnels de l'agent et les objectifs normatifs est justifiée dans [Conte et al., 1998] par la différence de motivation à l'origine des objectifs. Selon les priorités de l'agent, s'il priorise les normes, ou ses objectifs personnels. Par ailleurs, les objectifs normatifs disparaissent dès que la norme n'est plus applicable, tandis que les objectifs personnels ne sont supprimés que quand elles ne peuvent plus être réalisées. Les principes des architectures délibératives normatives sont abordés dans [Castelfranchi et al., 2000] où l'on démarquera les principes suivants :

1) Les normes doivent impacter d'une manière ou d'une autre sur le comportement des agents;
2) Les normes ne peuvent pas être des contraintes fixes et irrévocables sur les objectifs de l'agent, i.e. la violation de la norme doit être possible;
3) L'agent doit pouvoir reconnaître les normes qui existent;
4) L'agent doit pouvoir délibérément prendre en compte une norme dans son comportement, et délibérément violer une norme.
5) Adopter une norme ne signifie pas l'accomplir, mais plutôt de générer des objectifs et des plans pour accomplir la norme, et de reconnaître qu'elles s'appliquent à l'agent, mais l'adoption ne garantit pas que les objectifs ainsi générés seront priorisés par rapport aux autres objectifs déjà existants;

Par rapport à l'architecture délibérative classique, e.g. l'architecture BDI, les normes ont ainsi un impact sur le comportement des agents de par :

- la génération d'objectifs, c'est-à-dire des désirs, qui décrivent ce que l'agent doit faire, et non ce qu'il veut faire;
- la sélection d'objectifs, c'est-à-dire la transformation des désirs en intentions, car elles fournissent des critères de préférence entre les différents objectifs de l'agent, elles poussent l'agent à préférer un objectif par rapport à un autre;

- La génération de plans et la sélection de plans : elles peuvent préconiser quelles actions inclure ou ne pas inclure dans le plan et peuvent également fournir des critères sur quel plan est meilleur qu'un autre.

Globalement, plusieurs travaux se focalisent sur la notion d'objectif normatif pour pouvoir permettre à l'agent de décider s'il compte prendre en compte ou violer une norme [Conte & Castelfranchi, 1995], cependant, étant donné que les objectifs peuvent être en conflits entre eux (qu'ils soient issus des normes ou non), la question devient alors sur comment l'agent va gérer ces potentiels conflits entre ses objectifs, y compris ceux issus des normes. Nous allons présenter quelques architectures d'agent normatives qui adhèrent à cette stratégie et déterminer leur mécanisme pour gérer les conflits potentiels entre les différents objectifs, notamment le typage des agents, ou bien la notion de motivation.

1.4.3.1. BOID

BOID (*Belief Obligations Intentions Desires*) est une architecture normative proposée par [Broersen et al, 2001] qui adhère à cette adoption d'objectifs normatifs. Elle étend l'architecture BDI avec des normes, en incluant le concept mental d'obligation. BOID distingue ainsi deux types d'objectifs : les objectifs personnels internes de l'agent (ses désirs) et les objectifs sociaux externes issus des normes sociales.

Au vu des conflits potentiels entre les différents objectifs de l'agent qui sont distingués entre 1) ses objectifs personnels qui résultent de ses désirs et de ses intentions, et 2) les objectifs issus des normes, BOID repose sur le type de l'agent pour décider si les intentions priment sur les obligations, ou si inversement, les obligations sont toujours prioritaires quand les deux objectifs sont incompatibles. Le type de l'agent définit donc une hiérarchisation partielle ou totale entre les obligations, les intentions, les désirs, et les croyances. Les agents dans BOID sont donc catégorisés en plusieurs types :

- *Égoïstes* si ses désirs priment sur ses obligations;
- *Sociaux* si ses obligations priment sur ses désirs;
- *Stable* si ses intentions priment sur ses désirs et ses obligations.

Il est également possible d'affecter plusieurs types à un seul agent, par exemple il est possible qu'un agent soit *stable* et *social*. Vu qu'il y a quatre (04) composants mentaux à arranger par priorité, on aura donc 24 ordres possibles. Cependant, les croyances de l'agent ne peuvent être guidées par les autres composants mentaux, car c'est ce qui s'appelle littéralement des vœux pieux, où l'agent prend ses désirs pour une réalité. Nous n'aurons finalement que 6 types d'agents possibles avec B qui prime toujours sur tous les autres composants mentaux : BOID, BODI, BDOI, BIOD, et BIDO (cf Figure 13).

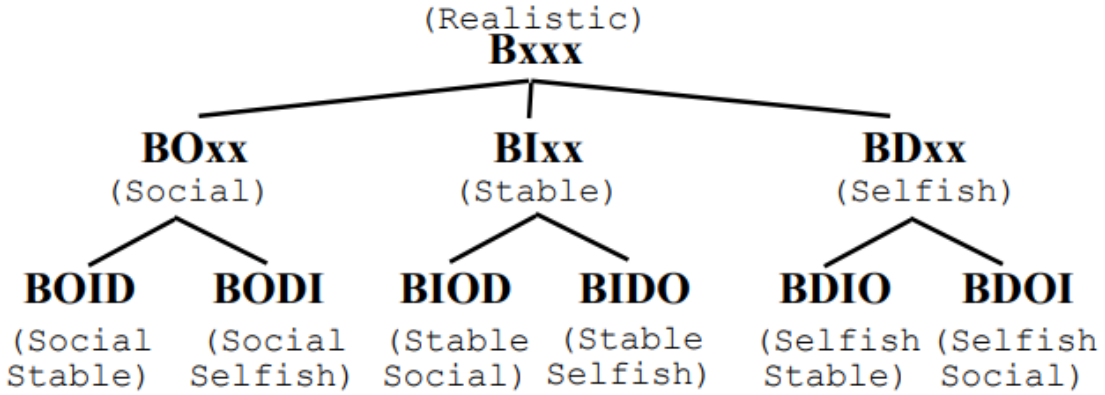


Figure 13: Arborescence des types d'agents possibles dans BOID

Dans la pratique, il est possible que l'agent décide d'exécuter des actions qui vont à l'encontre de son type. Plus loin, l'implémentation de l'architecture BOID se fait généralement à l'aide d'une bibliothèque de plans pour simplifier leurs implémentations, lorsqu'un objectif est choisi, qu'il soit issu d'une norme ou non, un plan solution est choisi (s'il y en a).

Étant une des extensions les plus populaires des architectures classiques, BOID se base donc sur un typage des agents pour décider quel objectif accomplir entre les objectifs normatifs. Dans les perspectives de [Broersen et al., 2001], on remarquera cependant que BOID, dans sa version originale, ne dispose pas encore d'un module pour planifier les normes et les objectifs que l'agent décide d'accomplir.

1.4.3.2. Lopez & Marquez

Les travaux de [Lopez & Marquez, 2004] se basent sur la notion de motivation d'importance des objectifs pour permettre aux agents d'exprimer des préférences entre ses objectifs, y compris les objectifs que les normes imposent d'accomplir dans une architecture normative abstraite.

Avant de pouvoir tirer les contributions des auteurs dans la prise en compte des normes, il est nécessaire de comprendre la représentation des normes utilisée, et de la rapporter à la grammaire institutionnelle que nous avons abordée sur les normes, qui est présentée sur la Figure 14.

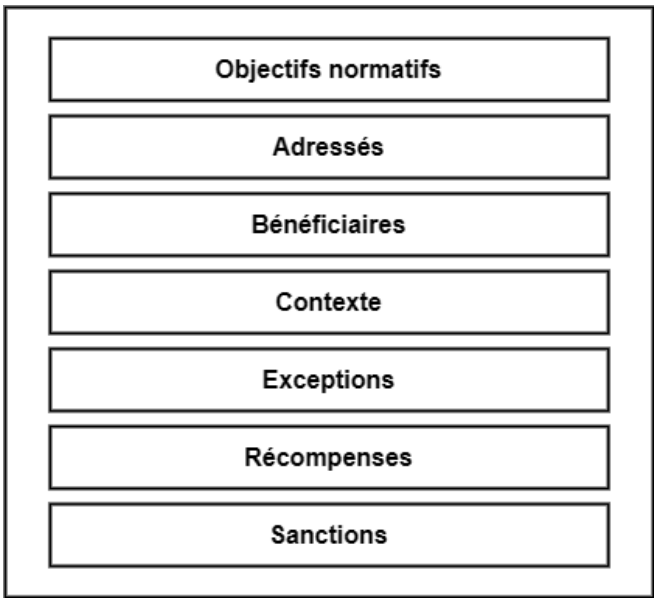


Figure 14: Structure d'une norme selon [Lopez & Marquez, 2004]

Les normes ont donc selon la Figure 14 :

1) Des objectifs normatifs qui décrivent ce qui doit être fait (obligations), ou les comportements qui doivent être inhibés (interdictions), ils correspondent aux objectifs de la norme (aIm) dans la grammaire ADICO;
2) Les adressés qui sont les agents chargés de satisfaire la norme et qui correspondent aux attributs (A) dans la grammaire institutionnelle ADICO;
3) Les bénéficiaires qui sont les agents qui bénéficient de l'accomplissement de la norme, pour déterminer pour qui doit-on respecter telle norme;
4) Le contexte qui décrit dans quelles conditions les normes sont applicables, à l'instar de la condition (C) dans ADICO;

5) Les exceptions qui décrivent les conditions dans lesquelles les agents ne subissent pas les sanctions en cas de violation de la norme;
6) Les récompenses et les sanctions, conformément à la partie otherwise (O) d'ADICO.

Les modalités déontiques traitées sont l'obligation et l'interdiction. Les auteurs se basent sur la comparaison des avantages et désavantages des normes dans le but de se décider à la rejoindre ou la quitter : les sanctions et les récompenses sont donc centrales pour adopter ou non une norme. La question devient alors :

- Est-ce qu'une récompense aide l'agent dans ses objectifs ?
- Est-ce qu'une sanction est en conflit avec un des objectifs de l'agent.

Les impacts des normes, à travers les objectifs qu'elles génèrent sur le comportement des agents dans [Lopez, 2003] se divisent en deux sous-processus : 1) la délibération des normes où l'agent décide s'il compte violer ou prendre en compte la norme qui lui est applicable, et 2) la prise en compte des normes où l'agent met à jour ses objectifs et l'importance des nouveaux objectifs ajoutés.

Durant la délibération des normes, l'agent évalue 1) les objectifs qui peuvent être obstrués par la prise en compte des objectifs normatifs, 2) les objectifs qui bénéficient de la prise en compte des récompenses des normes, et 3) les dégâts que peuvent causer les sanctions des normes. Pour cela, les normes applicables sont divisées en normes non-conflictuelles, et normes conflictuelles pour déterminer quelle stratégie sera appliquée pour décider si la norme est prise en compte ou violée.

Formellement, un agent normatif classifie les normes en plusieurs catégories : 1) les normes actives, 2) les normes intentionnelles, et 3) les normes rejetées; et définit trois types d'objectifs supplémentaires en plus de ceux de l'agent : 1) les objectifs de la norme, 2) les objectifs de la sanction de la norme; et 3) les objectifs de la récompense de la norme. Trois prédicats sont également définis pour désigner les relations entre les différents objectifs et les normes:

1. *hinders(g1, g2)* qui désigne que l'objectif *g1* nuit à l'objectif *g2*;
2. *benefits(g1, g2)* qui désigne que l'accomplissement d'un objectif *g1* contribue à l'accomplissement d'un autre objectif *g2*;

3. *conflicts(norm) :* est un prédicat qui est vrai si et seulement si une norme est en conflit (une relation de type *"hinders"*) avec un des objectifs actuels de l'agent.

La délibération des normes et leur prise en compte dans cette approche présente les étapes suivantes :

1) Récupération de l'ensemble des normes applicables à l'agent;
2) Extraction des objectifs normatifs, i.e. des objectifs liés aux sanctions et des objectifs liés aux récompenses;
3) Classification des normes qui sont bénéfiques ou nuisibles de par leurs objectifs à l'agent;
4) Pour délibérer si une nouvelle norme η sera prise en compte ou non, elle sera ajoutée tout d'abord dans l'ensemble des normes intentionnelles, tandis que l'ensemble des normes rejetées reste la même;
5) L'ensemble des objectifs actuels est mis à jour de sorte à intégrer les objectifs de la norme en question : les objectifs de la norme, de ses sanctions et de ses récompenses;
6) Tous les objectifs $o_{beneficiary}$ tel que *benefits*(η, $o_{beneficiary}$) sont retirés car ils sont accomplis par d'autres moyens, i.e. l'accomplissement d'autres objectifs;
7) Les objectifs qui sont obstrués par la nouvelle norme, i.e. tous les objectifs $o_{hindered}$ tels que $hinders(\eta, o_{hindered})$ est vrai, seront suspendus.

La Figure 15 illustre ce processus avec un diagramme de Venn[5], et comment [Lopez, 2003] utilise les termes "(non) conflicting norms" pour désigner les normes (non) conflictuelles avec les objectifs existants; ainsi que les termes *intendedNorms* et *rejectedNorms* pour décrire les normes qui sont prises en compte et qui ne seront pas prises en compte dans le comportement de l'agent.

[5] Un diagramme de Venn est un diagramme qui montre toutes les relations logiques possibles dans une collection finie de différents ensemble. Ici, elles montrent comment les normes actives (adopted norms) peuvent être catégorisées.

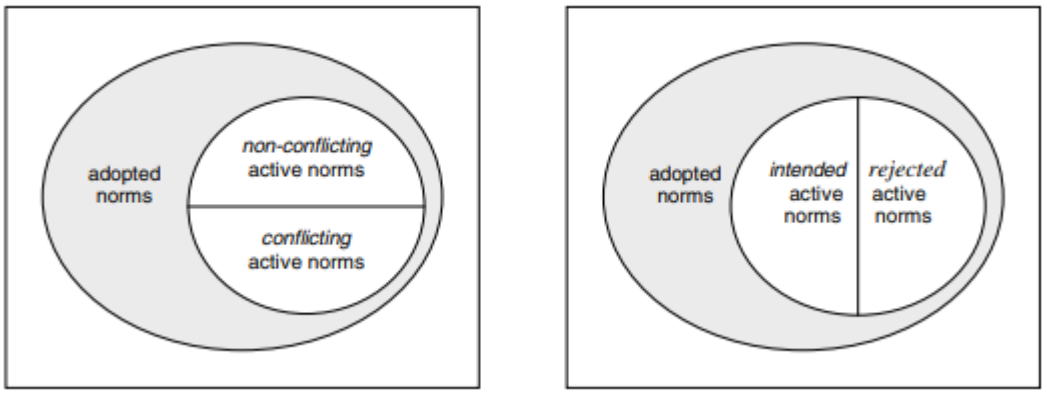


Figure 15: Division des normes à gauche avant la délibération des normes, et à droite après la délibération des normes [Lopez, 2003]

Pour prendre en compte les normes, plusieurs stratégies peuvent être choisies par le modélisateur:

a) *Social*: qui priorise les objectifs normatifs aux objectifs personnels de l'agent. Les agents sociaux ne subissent donc jamais de sanctions, et reçoivent le maximum de récompenses.;
b) *Pressurisé*: qui considère les effets des sanctions sur ses objectifs personnels et qui agit en conséquence
c) *Opportuniste*: qui considère les effets des récompenses sur ses objectifs.
d) *Craintif* : qui adopte une norme si celle-ci contient une sanction;
e) *Avide*: qui adopte une norme si celle-ci contient une récompense;
f) *Rebelle* : qui rejette tout simplement toutes les normes.

Formellement, nous noterons NC (*Non comply*) la conclusion que l'agent va violer la norme, et inversement, nous noterons C *(Comply)* la conclusion que l'agent va respecter la norme.

Social. L'adoption des normes de l'agent social se caractérise par un ensemble vide de normes rejetées, soit :

$$rejectednorms = \emptyset.$$

Pour les agents pressurisés, une norme qui n'est pas en conflit avec les objectifs de l'agent est adoptée quand ses sanctions n'obstruent pas d'autres objectifs existants. Formellement :

$$\neg conflicting(new?) \ \wedge \ hindered(goals, new?, punishments, hinders) \ \neq \ \emptyset \Rightarrow C$$

Dans le cas d'une norme non conflictuelle et qui n'obstrue pas les objectifs existants, la norme est rejetée.

$$\neg conflicting(new?) \ \wedge \ hindered(goals, new\,?, punishment, hinders) \ = \ \emptyset \Rightarrow NC$$

Sinon, si la norme est conflictuelle, l'agent va prendre en compte la norme aux dépens des objectifs actuels si les objectifs obstrués par les sanctions de la norme sont plus importants que l'ensemble des objectifs obstrués par les objectifs normatifs. Formellement, pour une nouvelle norme conflictuelle noté $new?$

- Soit : $gs_1 \ = \ hindered(goals, new?\ punishment, hinders)$
- Soit : $gs_2 \ = \ hindered(goals, new?\ normativegoals, hinders)$
- $importance\ motivations\ gs_1 \ > \ importance\ motivations\ gs_2 \ \Rightarrow \ C$
- $importance\ motivations\ gs_1 \ < \ importance\ motivations\ gs_2 \ \Rightarrow \ NC$

Opportuniste. À l'instar d'un agent pressurisé, l'agent prend en compte les normes non-conflictuelles seulement si celles-ci contribuent à l'accomplissement d'un de leurs objectifs. Formellement, pour une nouvelle norme $new?$ non conflictuelle :

$$benefited(goals, new?\ .rewards, benefits) \ \neq \ \emptyset \Rightarrow C$$

$$benefited(goals, new?\ .rewards, benefits) \ = \ \emptyset \Rightarrow NC$$

Pour ce qui est des normes conflictuelles, la motivation est le facteur qui détermine la décision de l'agent : les normes conflictuelles sont prises en compte si ses récompenses sont plus importantes que les objectifs qui sont obstrués par la norme. Pour toute norme conflictuelle ? new:

Soit :

- $gs_1 = hindered(goals, new?\ .rewards, benefits)$
- $gs_2 = hindered(goals, new?\ normativegoals, hinders)$
- $importance\ motivations\ gs_1 > importance\ motivations\ gs_2 \Rightarrow C$
- $importance\ motivations\ gs_1 \leq importance\ motivations\ gs_2 \Rightarrow NC$

Craintif. Pour cette stratégie, l'existence de sanctions suffit pour prendre en compte la norme.

$$\forall n : rejectednorms,\ n.punishments = \emptyset$$

$$\forall n : intendednorms,\ n.punishments \neq \emptyset$$

Avide. Inversement, pour les agents avides, l'existence de récompenses suffit pour prendre en compte la norme sans délibération.

$$\forall n : rejectednorms,\ n.rewards = \emptyset$$

$$\forall n : intendednorms,\ n.rewards \neq \emptyset$$

Rebelle. Les agents rebelles rejettent tout simplement toutes les normes.

$$intendednorms = \emptyset$$

Si les travaux de [Lopez, 2003; Lopez & Marquez, 2004] offrent plus de stratégies pour gérer les conflits entre les objectifs, y compris les objectifs normatifs, avec les stratégies : rebelle, avide, craintif, opportuniste, social, et pressurisé. Toutefois, cette stratégie reste statique puisque l'agent ne peut pas changer de type au cours de la simulation. De par son abstraction, cette contribution se focalise sur quelles normes choisir comme intention, sans détailler quel plan prédéfini y associer ou comment construire un nouveau plan satisfaisant la norme.

1.4.3.3. Normative AgentSpeak(L)

Normative AgentSpeak(L) de [Meneguzzi & Luck, 2009], propose des méta-actions pour scanner la bibliothèque de plans prédéfinis de l'agent, et la modifier pour retirer l'ensemble des plans qui violent les normes que l'agent prend en compte. Les normes traitées dans Normative AgentSpeak sont les obligations et les interdictions. Ils proposent de modifier la bibliothèque de plans selon la modalité déontique des normes:

1) pour les interdictions : les plans qui violent les normes sont temporairement retirées de la bibliothèque de plans tant que l'interdiction est applicable;
2) pour les obligations : de nouveaux plans sont ajoutés grâce à un planificateur pour permettre aux agents d'accomplir l'obligation [Meneguzzi & Luck, 2007].

Le planificateur utilisé formule un problème de planification qui sera traité par un planificateur basé sur STRIPS de [Fikes & Nilsson, 1971] que nous aborderons plus en détails en section 1.6.1. Dans cette optique, deux conversions sont effectuées :

1) Les concepts de AgentSpeak sont transformés en concepts STRIPS pour pouvoir formuler le problème de planification : les plans de la bibliothèque de plans prédéfinis sont convertis en actions avec des préconditions et des postconditions, les objectifs seront utilisés pour décrire un état final, les croyances de l'agent vont constituer la spécification de la situation initiale. Une revue détaillée de STRIPS, et de ce qu'est un problème de planification est présentée dans 1.6.;
2) Une fois qu'un plan solution est trouvé, les concepts de STRIPS du planificateur sont traduits en AgentSpeak de nouveau, et ajoutés à la librairie de plans puis aux intentions de l'agent.

Le planificateur peut donc déduire de la librairie de plans, des croyances de l'agent, et de ses objectifs de nouveaux plans pour prendre en compte les obligations. Dans le cas où un plan prédéfini échoue, le planificateur est également sollicité. Ces travaux se focalisent surtout sur comment adapter le comportement des agents aux normes acceptées par l'agent. Les auteurs assument que l'agent essaye toujours de se conformer aux normes, et ne détaillent pas comment ils peuvent décider si les normes sont acceptées ou rejetées.

1.4.3.4. EMIL-A

EMIL-A issu du projet EMIL (EMergence In the Loop Architecture) de [Andrighetto et al., 2007; Andrighetto et al. 2010], est une architecture normative théorique qui se base sur l'utilité attendue si l'agent prend en compte ou viole les normes.

Dans EMIL-A, l'adoption d'une norme en tant qu'objectif ou intention se base sur un critère d'utilité: la saillance des normes[6] [Andrighetto et al. 2010]. Cette saillance de la norme est utilisée comme critère de prise en compte ou de violation des normes: plus une norme est saillante, plus grande est la probabilité que l'agent la prenne en compte.

Les recherches sur EMIL-A ont adopté une approche une approche "pourquoi pas" : ils ont une préférence à prendre en compte les normes s'il n'a pas de raison de les violer. Par exemple, quand sa prise en compte ne dépasse pas son coût. EMIL-A dispose de plusieurs extensions tel que EMIL-I-A [Andrighetto et al., 2012] pour l'internalisation des normes, EMIL-S [Lotzmann et al, 2009] pour implémenter les concepts de EMIL-A, mais se focalisent plus sur l'émergence des normes que la génération de comportements complexes.

1.4.3.5. n-BDI

n-BDI est une extension de l'architecture BDI en une architecture normative et multi-contexte graduée et normative initiée par [Criado et al, 2010]. Parmi ses nombreuses stratégies de prise en compte des normes, n-BDI propose de modéliser les objectifs sociaux comme désirs de l'agent, qui seront délibérés en fonction d'une valeur de priorité.

Les contextes désignent globalement des modules cognitifs avec leur propre langage, axiomes et règles qui offrent une logique modale pour graduer les croyances, les désirs, et les intentions.

Dans n-BDI, une norme n est définie comme $n = < D, C, A, E, S, R >$ où :

- D $\in$ {O, F}, est l'opérateur déontique.Seules les obligations (O) et les interdictions (F), qui imposent des contraintes aux comportements des agents, ont été prises en compte ; les permissions n'ont pas été prises en compte, car elles sont définies comme des opérateurs qui invalident l'activation d'obligations ou d'interdictions ;

[6] La saillance est récemment défini par Andrighetto comme étant "la perception du degré d'importance et de force d'une norme"

- C est un ensemble de prédicats qui représente la condition normative qui doit être exécutée dans le cas des obligations, ou qui doit être évitée dans le cas des normes d'interdiction ;
- A, E sont des prédicats du premier ordre qui déterminent les conditions d'activation et d'expiration de la norme, respectivement ;
- S, R sont des expressions qui décrivent les actions (sanctions et récompenses) qui seront menées en cas de violation ou d'accomplissement de la norme, respectivement.

Pour la prise en compte des normes, n-BDI permet la mise en œuvre des stratégies classiques :

1) la stratégie automatique qui va accepter toutes les normes;
2) la stratégie rebelle qui va rejeter toutes les normes;
3) la stratégie craintive qui va accepter toutes les normes ayant une sanction (qu'elle soit bénéfique ou non);
4) la stratégie avide qui va accepter toutes les normes ayant une récompense, sans tenir compte de son utilité.

Des stratégies motivationnelles plus complexes pour la prise en compte des normes sont également disponibles à l'aide d'une fonction d'utilité $\boldsymbol{\delta}_i$ qui mesure la *désirabilité* $\boldsymbol{\delta}_i^+$ ou *l'indésirabilité* $\boldsymbol{\delta}_i^-$ de l'application d'un composant i de la norme : son application, ses sanctions ou bien ses récompenses. En comparant les fonctions d'utilités, nous pouvons obtenir les stratégies suivantes :

1) la stratégie égoïste acceptera uniquement les normes qui contribuent à ses objectifs à travers ses récompenses;
2) la stratégie de la pression si la sanction est beaucoup plus pénalisante que l'application de la norme;
3) la stratégie opportuniste acceptera une norme si la récompense de la norme est plus importante que la pénalisation de son implication.

Une autre famille de stratégies complexes pour prendre en compte les normes sont les stratégies de coopération corrélées aux normes où :

1) l'agent coopératif qui va respecter une norme si elle est bénéfique pour la société, avec une incorporation des objectifs sociaux en tant que désirs des agents;
2) l'agent bénévole qui va respecter toute norme bénéfique pour un autre agent qu'il veut favoriser.

Sur la base de ces stratégies, *n-BDI* peut construire des stratégies plus complexes :

1) la stratégie mixte qui va respecter une norme si l'ensemble des bénéfices de son application est supérieur à l'ensemble des pénalités de sa violation, c'est-à-dire, si la somme des intérêts issu de l'application de la norme, des récompenses et de *l'indésirabilité* de la sanction, surpasse la somme de la désirabilité de la sanction, et de *l'indésirabilité* de la norme, et de ses récompenses.
2) la stratégie mixte pondérée qui est exactement similaire à la stratégie mixte, mais qui va pondérer la désirabilité et *l'indésirabilté* des sanctions et des récompenses par leurs probabilités.

Malgré la mise en place de ces stratégies pour choisir quel objectif accomplir en fonction de la priorité associée, n-BDI ne propose pas de stratégies pour gérer les conflits entre normes. De même, l'évaluation de la désirabilité ou de l'indésirabilité d'une norme n'est pas assez explicite, puisqu'aucune méthode automatique ne permet de les estimer, comme le mentionne [Viana et al., 2015], et ne mentionne pas les rôles des normes sur comment les agents construisent leurs plans [Fagundes et al., 2016].

1.4.3.6. ANA

ANA ou Autonomous Normative Agents [dos Santos Neto et al., 2012] réutilise les travaux de [Lopez & Marquez, 2004] sur les types d'agents et les objectifs conflictuels, toujours en définissant une norme comme étant constituée par :

1. Les adressés de la norme;
2. La modalité déontique qui peut être une obligation ou une interdiction (les permissions ne sont pas prises en compte);
3. Le contexte d'activation et de désactivation;
4. L'état que la norme vise à accomplir;

5. Les actions et les récompenses de la norme.

ANA ajoute à l'architecture BDI plusieurs extensions notamment par un composant appelé *NormSelector*, qui est chargé d'analyser et de choisir les normes qu'il compte convertir en intentions. Pour cela, le *NormSelector* doit résoudre les conflits potentiels entre normes.

Résolution des conflits. Un conflit se présente quand une obligation et une interdiction spécifient le même état, i.e. ont le même objectif; Si la motivation de l'agent pour l'obligation est plus forte, elle sera choisie pour être accomplie, et l'interdiction sera violée, et vice-versa. En cas d'égalité, n'importe laquelle des deux peut être choisie.

La motivation d'un agent pour accomplir ou violer une norme est calculée par les sanctions et les récompenses, pour la motivation à accomplir une norme, elle se calcule par la somme de l'influence de l'opérateur déontique sur l'état cible et de la motivation des récompenses. Inversement, la motivation pour violer une norme est la motivation à recevoir les sanctions.

L'influence de l'opérateur déontique est calculée comme suit : i) si c'est une obligation: elle est positive si l'agent souhaite atteindre cet objectif, sinon, elle est négative puisque la norme accomplit un objectif que l'agent ne veut pas; ii) si c'est une interdiction, elle est positive si l'agent ne souhaite pas accomplir l'état interdit, sinon, elle est négative puisque l'agent se voit interdire un état qu'il souhaite accomplir.

Il est important de noter que pour faire ce calcul, la motivation de l'agent sur chaque objectif doit être quantifiée avec des entiers positifs ou négatifs au préalable.

Choix des normes. Lorsque les normes n'ont aucun conflit, l'agent peut choisir entre satisfaire les obligations et les interdictions, ou les violer. Le calcul se fait en comparant selon la modalité déontique :

- Pour les obligations : si la somme de la motivation de l'agent pour l'état cible de la norme et sa motivation pour les récompenses est plus grande que sa motivation pour les sanctions, alors l'agent accomplit l'obligation, sinon, il viole l'obligation;
- Pour les interdictions : on vérifie si la somme de la motivation de l'agent à recevoir les récompenses moins la motivation de l'agent de ne pas satisfaire l'état cible, est

plus grande que sa motivation à recevoir les sanctions; si oui, l'agent accomplit l'interdiction, sinon, il viole l'interdiction.

Prise en compte des normes. Des objectifs sont produits et les motivations de l'agent par rapport à ses objectifs sont mises à jour de sorte à prendre en compte les récompenses et les sanctions obtenues, et des décisions prises. Ceci permet à l'agent de poursuivre sur sa lancée et d'incrémenter ou décrémenter ses motivations dynamiques qui changent au fil de ses décisions. La méthode de mise à jour est décrite plus en détails dans [dos Santos Neto et al., 2012].

Comparé aux autres approches d'architectures normatives telle que BOID de [Broersen et al., 2001] ou [Lopez & Marquez, 2004] qui proposent des types d'agents (égoïstes, sociaux etc.), l'approche d'ANA offre une solution quantifiable dynamique pour que l'agent puisse i) résoudre un conflit de normes différemment au fil de la simulation, ii) décider entre prendre en compte ou violer une norme.

La notion de motivation dans ANA basée sur les sanctions et les récompenses qui se met à jour à chaque violation ou prise en compte d'une norme est également une innovation par rapport aux travaux de [Lopez & Marquez, 2004]. Toutefois, une quantification initiale des objectifs rattachés à chaque norme est nécessaire pour rendre ANA fonctionnel, et les auteurs s'arrêtent au fait de choisir quel objectif accomplir entre ceux des normes et ceux de l'agent.

1.4.4. Synthèse sur les architectures d'agent normatives

Plusieurs architectures normatives de la littérature se focalisent surtout sur la génération d'objectifs normatifs comme mécanisme de prise en compte des normes. Ces extensions partent du prérequis que l'agent doit être un agent délibératif pour prendre en compte les normes, comme le décrit [Castelfranchi et al., 2000].

La décision de prendre en compte les normes ou non pour un agent donné se résume à déterminer si ces normes sont en conflit avec les objectifs actuels des agents, et si elles ne le sont pas, sont-elles prioritaires? Pour y répondre, différents mécanismes pour instaurer une

préférence sont introduits pour permettre aux agents de calculer s'il est plus avantageux de prendre compte ou de violer la norme, notamment :

a) l'ajout de notions de motivation de l'agent comme les travaux sur ANA, qui se rapproche également des fonctions d'utilité comme pour l'architecture EMIL-A,
b) ou b) grâce à un typage des agents permettant d'adopter une stratégie fixe pour choisir entre les normes et les objectifs personnels de l'agent, e.g avec BOID, ou les travaux de [Lopez & Marquez, 2004];

Les architectures normatives doivent prendre en compte plusieurs problèmes en plus de l'architecture BDI classique :

1) Décider d'adopter ou non une norme lorsque l'agent sait qu'elle lui est applicable;
2) Vérifier si la norme adoptée est appliquée ou violée;
3) Résoudre les conflits entre normes au cours de la génération de comportement, et potentiellement les conflits entre les objectifs de l'agent et les normes;
4) Décider quelles normes seront finalement violées et lesquelles seront prises en compte

Mais à l'instar des architectures d'agent classiques, les architectures normatives théoriques sont limitées car la plupart des approches normatives ne spécifient pas comment l'agent trouve un plan pour atteindre un objectif normatif. Similairement aux implémentations des architectures classiques, elles se basent sur une bibliothèque de plans pour atteindre un objectif donné, y compris pour les objectifs normatifs. Cette implémentation procédurale permet d'obtenir des implémentations efficaces en termes de calcul, mais empêche l'agent de trouver d'autres alternatives pour satisfaire ses objectifs et ceux des normes.

Ces tendances se confirment par les extraits de propos sur non seulement les architectures d'agent normatives, mais plus généralement sur l'ensemble des architectures d'agent :

- Dans [Andrighetto & Conte, 2012, p. 375] les auteurs d'EMIL-A mentionnent: "Il ne s'agit pas ici de traiter en détail de l'activité du planificateur. Nous nous contenterons

d'aborder la question du moment où un objectif normatif, une fois formé, a le plus de chances d'être exécuté" [7].

- Dans [Panagiotidi & Vazquez-Salceda, 2011] les auteurs formulent que beaucoup de théories des normes, leur prise en compte dans leur processus décisionnel sont abordées dans la littérature, mais peu sont rendus opérationnelles, ce qui est confirmé dans [Neumann, 2010].

En reprenant la synthèse de [Castelfranchi et al., 2000], les agents qui traitent des normes du début des années 2000 souffrent de plusieurs désavantages :

1) L'utilité d'un agent sur les objectifs d'une norme et ses récompenses est fixe, excepté pour des architectures plus récentes telles que ANA publiée en 2012;
2) Aucun de ces papiers ne donne de directives claires et unifiées de "comment il faut assigner ces valeurs quantitatives?"

De plus, une grande majorité des architectures classiques et des architectures normatives plus récentes se basent sur une bibliothèque de plans prédéfinis, ce qui limite l'autonomie des agents dans un environnement qui se veut ouvert et dynamique. La question de l'autonomie de l'agent pose également problème, puisque l'agent doit pouvoir déterminer comment accomplir son objectif, sous la contrainte de normes, comment corriger son comportement pour se conformer à une nouvelle norme automatiquement dans un environnement ouvert, qui implique de nouvelles normes.

En termes de modalité déontique, plusieurs architectures également ne traitent pas entièrement de toutes les modalités possibles, notamment la permission qui est parfois omise. De plus, les notions d'institution, d'organisation, de rôle ne sont pas toujours reprises dans les différentes approches évoquées dans cet état de l'art, marquant ainsi la disparité de concepts entre les architectures normatives et les normes selon l'économie institutionnelle, dont la grammaire des institutions proposée par [Crawford & Ostrom, 1995].

Vu ces limitations et ces abstractions des architectures d'agent normatives, il est nécessaire de voir comment les architectures rendues opérationnelles à travers des langages orientés agents et/ou des plateformes sont implémentées.

[7] De l'anglais original : " This is not the forum for a detailed treatment of the planner's activity. We limit ourselves to addressing the question of when a normative goal, once formed, is most likely to be executed"

1.5. Implémentation des normes dans les SMA

Dans cette section sont présentés les différents langages et outils opérationnels pour représenter les organisations, les institutions, et les normes. Pour chaque section, nous déterminerons également comment elles produisent le comportement adéquat pour atteindre les objectifs de l'agent.

1.5.1. MOISE

MOISE (Model of Organization for Multi-Agent Systems) est un langage de modélisation d'organisations, qui spécifie également comment elles interagissent avec les agents [Hannoun et al., 1999]. La représentation des organisations dans MOISE se base sur trois dimensions :

1) La dimension structurelle pour définir les groupes, les rôles et les liens entre les rôles;
2) La dimension fonctionnelle pour définir les objectifs d'une organisation et des missions pour chaque rôle;
3) La dimension normative, avec la modalité déontique qui va définir ce qui est obligatoire, interdit, et permis pour chaque rôle.

La dimension structurelle, notée *SS* est définie dans [Balbo et al., 2010] comme étant un tuple $SS = < \mathcal{R}, \sqsubset, rg >$ où $\mathcal{R}$ est l'ensemble des identifiants des rôles, $\sqsubset$ les relations d'héritages sur les rôles, et rg une spécification du groupe racine de l'organisation. Elle permet de définir les rôles de compatibilité entre rôles, les cardinalités maximales et minimales d'agents pouvant jouer un rôle au sein du groupe, et les liens entre les groupes et les sous-groupes.

La dimension fonctionnelle *FS* est un triplet $FS = < M, G, S >$ où M est un ensemble de missions (un groupement d'objectifs individuels ou collectifs cohérents) auxquelles l'agent s'engage à accomplir à travers les rôles qu'il adopte (similairement aux intentions BDI); G est l'ensemble des objectifs collectifs à satisfaire. S est l'ensemble des schémas sociaux, qui structurent les objectifs en plans.

La dimension normative *NS* est le composant chargée de décrire un ensemble de normes, chacune décrite par le tuple suivant

$$norm = \ < id,\ c,\ \rho,\ dm,\ m >$$

où : *id* un identifiant unique de la norme, *c* la condition (d'applicabilité) de la norme, ρ est le rôle sur lequel la modalité déontique s'applique *dm* (obligation ou permission), *m* est la mission que la norme impose. Une norme peut ainsi s'exprimer comme suit : "lorsque *c* est vérifiée, les agents jouant le rôle ρ ont *dm* de s'engager sur la mission *m*".

Les seules modalités déontiques des normes sont les obligations et les permissions. Cependant par défaut, s'il n'existe aucune permission ou obligation sur une paire mission-rôle, il est alors interdit pour l'agent de s'engager sur la mission. Par rapport à ce que nous avons défini comme étant une institution et une organisation dans la section 1.1.4.Synthèse sur les normes, *MOISE* définit ce que nous entendons par institution par une structure organisationnelle (OS), et ce que nous entendons par une organisation par une entité organisationnelle (OE) qui rattache un ensemble d'agents à cette structure.

Par rapport à notre sujet d'intérêt qui est la prise en compte des normes et de comment celle- ci est prise en compte, celle-ci se déroule dans le FS qui décrit la dimension fonctionnelle, i.e. comment les objectifs sont décomposés en plans et distribués aux agents sous la forme de missions. Celle-ci est détaillée dans MOISE+ [Hübner et al., 2002] qui introduit les concepts pour des plans globaux pré-validés pour accomplir les objectifs d'une mission, qui vont ensuite être divisés en sous-objectifs que les agents peuvent s'engager à accomplir. Une notion de préférence entre missions est également présentée.

Malgré tout, *MOISE* et ses variantes telles que *MOISE+* ne prennent pas en compte le fait que les agents peuvent avoir leurs objectifs, et potentiellement d'autres objectifs sociaux, et de facto, ils ne couvrent pas comment un agent va appliquer ou violer les normes dans son comportement. Dans ce sens, les travaux sur *MOISE*, *MOISE+* et plus loin $MOISE^{inst}$ ne définissent pas de système de planification ou d'exécution, et s'orientent plus vers des institutions électroniques où l'objectif est plus de pouvoir décrire des règles pour orchestrer le comportement de différents agents, et moins de comprendre comment celle-ci décide d'agir en conciliant les normes à ses objectifs personnels. L'autonomie sociale et l'autonomie en

termes de planification sous l'influence des normes n'est pas représentée dans MOISE comme le font remarquer des revues ultérieures [Maia & Sichman, 2020].

1.5.2. JaCaMo

JaCaMo (Jason CArtAgO Moise) [Boissier et al, 2013] est un framework de développement multi-agent, prenant en compte à la fois la dimension organisationnelle, la dimension de l'environnement, et la dimension de l'agent. La prise en compte de ces trois dimensions unifie: 1) Ja : la programmation des agents avec Jason [Bordini & Hübner, 2006], 2) Ca : la programmation de l'environnement avec CArtAgO [Ricci et al., 2011] , et 3) Mo : la programmation des organisations avec *Moise* [Hannoun et al., 1999; Hubner et al., 2002]. Le méta-modèle de JaCaMo est présenté sur la Figure 16 avec les trois dimensions traitées.

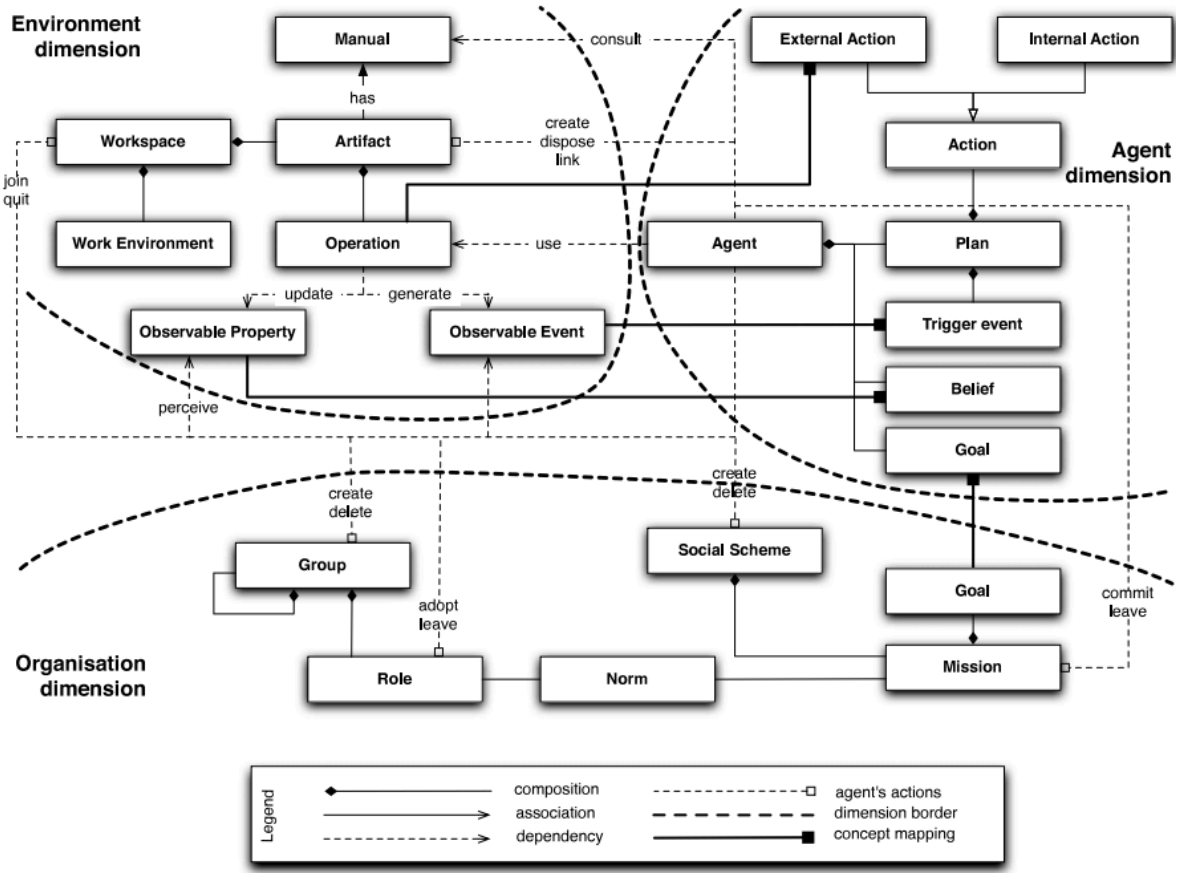


Figure 16: Méta-modèle de JaCaMo et des relations et correspondances entre les concepts des différentes dimensions, présenté dans [Boissier et al., 2013]

Pour rapporter les concepts de ce méta-modèle à la prise en compte des normes, nous nous intéresserons majoritairement à la dimension qui traite des organisations, et de l'agent. Le processus de perception des événements de l'environnement et la modification de ce dernier grâce à des artéfacts dépasse la portée du présent travail: nous nous intéressons principalement à ce que les agents font en réponse quand après avoir perçu ces structures organisationnelles, avec les mises à jour sur l'état de l'environnement et ses propres objectifs, pour choisir un comportement adapté et expliquer ce choix.

En termes de génération de comportements, ces relations permettent à JaCaMo d'enrichir le modèle standard des langages de programmation orienté-agent classiques avec :

1) Un répertoire d'actions dynamique qui permet à l'agent d'ajouter, et d'obtenir de nouvelles actions au lieu des actions définies statiquement de manière ad-hoc;
2) Une représentation d'action plus expressive grâce à leur correspondance aux opérations, permettant l'exécution d'actions concurrentes et des actions à long terme pour pouvoir synchroniser les actions de plusieurs agents;
3) Une représentation explicite de l'échec ou de la réussite d'une action plutôt que de vérifier l'existence de ses post-conditions dans la base de croyances à la fin de chaque exécution.

Pour choisir les plans d'actions permettant de réaliser un objectif, JaCaMo met à disposition un ensemble d'actions basiques prédéfinies pour éviter de les implémenter de manière ad hoc, par exemple l'adoption d'un rôle, l'interaction avec l'environnement. La dimension des organisations, qui est distincte de l'agent, fournit grâce à des artéfacts organisationnels des actions pour que l'agent puisse prendre part à l'organisation, i.e. ils définissent une ontologie, y compris pour les actions possibles dans l'organisation.

1.5.2.1. Représentation des normes

La représentation des normes est définie par les organisations dans MOISE. Une organisation dispose de :

- La spécification structurelle : avec l'ensemble des groupes et les différents rôles au sein de ces groupes.
- La spécification fonctionnelle : Il s'agit de la raison d'être de l'organisation (mission) et de ses objectifs généraux (buts). Le schéma social agit comme une feuille de route, décomposant les grands objectifs en étapes plus petites et réalisables. Ces étapes sont ensuite regroupées en missions spécifiques.
- La spécification normative : il s'agit des règles qui relient des rôles aux missions auxquelles elles contribuent avec une modalité déontique.

Pour la prise en compte des normes, les objectifs des organisations, i.e. les normes, sont transformés en objectifs pour des agents individuels, et ces derniers peuvent les adopter ou non. JaCaMo exprime ces objectifs sous la forme d'une obligation qu'il faut que l'agent satisfasse avant une échéance, l'agent peut non seulement choisir de l'adopter ou non, mais peut aussi choisir librement quel plan choisir pour satisfaire ces obligations, leur laissant ainsi leur autonomie.

1.5.2.2. Prise en compte des normes

Les obligations sont formulées sous la forme de "missions" dans JaCaMo, qui est un ensemble d'objectifs et de sous-objectifs que l'agent doit satisfaire. Les plans dans les implémentations de JaCaMo, sont pour la plupart, inspirés de JASON et AgentSpeak [Bordini & Hübner, 2005] sont implémentées sous la forme d'une bibliothèque de plans, et sont formulées sous la forme d'une règle.

$$event: context <- body$$

Event représente les événements qui peuvent entraîner l'exécution du plan, c'est-à-dire, les événements pour lesquels le plan peut être pertinent. *Context* qui est un ensemble de croyances qu'il faut remplir pour que le plan soit applicable. Body représente le corps du plan qui contient l'ensemble des actions, et des sous-objectifs à remplir quand le plan est exécuté. Par exemple, lorsque les croyances de l'agent changent, un événement est produit et sera pris en charge par les plans, si le plan est applicable, il peut être choisi.

JaCaMo n'a cependant pas de moyen pour créer un plan de zéro directement, mais offre le moyen de changer la bibliothèque de plans lors de la simulation à travers sa version web: JaCaMo Web [Amaral & Hubner, 2019]

1.5.3. JSAN

JSAN est un framework qui permet de prendre en compte les normes dans le processus décisionnel des agents [Viana et al., 2015] qui s'appuie sur la plateforme JASON [Bordini & Hübner, 2006] pour créer des agents BDI normatifs avec le langage AgentSpeak, et propose les stratégies : égoïste, rebelle et social pour que les agents prennent en compte les normes.

1.5.3.1. Représentation des normes

Pour comparer la représentation des normes dans JSAN, nous proposons une correspondance de cette représentation avec la grammaire institutionnelle ADICO de [Crawford & Ostrom, 1995] dans le Tableau 3.

Tableau 3: Les éléments d'une norme dans JSAN, comparés aux éléments de ADICO

Élément	Description	Équivalent ADICO
Addressee	Les agents ou les rôles qui sont chargés de respecter la norme	A
Activation	La condition d'applicabilité de la norme	C
Expiration	La condition d'expiration de la norme	Insatisfaction de C
Rewards	Les récompenses octroyées à l'agent pour avoir appliqué une norme.	O
Punishments	Les sanctions pour avoir violé une norme.	O
DeonticConcept	Indique si la norme spécifie une obligation, une permission ou une interdiction.	D
State	Décrit l'ensemble des états qui sont normalisés.	I

1.5.3.2. Stratégies de prise en compte des normes

Dans JSAN, les plans sont déjà prédéfinis dans une bibliothèque de plans à l'instar de son prédécesseur JASON. Pour prendre en compte les normes applicables à l'agent se base sur deux informations :

i) l'ensemble de ses objectifs qui seront en conflit avec les objectifs normatifs;

ii) l'ensemble des objectifs qui seront accomplis à travers les récompenses obtenues pour avoir complété l'objectif normatif.

En fonction de la stratégie choisie, le choix du plan à adopter une stratégie :

- Égoïste : qui priorise les normes ayant des récompenses bénéfiques pour les objectifs de l'agent;
- Social : qui priorise les objectifs normatifs par rapport à ses objectifs personnels;
- Rebelle : qui rejette les normes;

Les agents dans JSAN peuvent être définis avec des classes Java, notamment :

- La classe *NormativeAgent* en faisait une sous classe en implémentant la méthode execute pour décrire les algorithmes possibles pour exécuter les plans de l'agent;
- La classe *NormStrategy* permet de définir d'autres stratégies (ou plans) pour prendre en compte les normes en plus des stratégies classiques : égoïste, rebelle et social.

De manière générale, JSAN se base sur la notion de récompenses ou de sanctions pour choisir le plan approprié à la stratégie prédéfinie de l'agent parmi une bibliothèque d'actions prédéfinies, et met en œuvre trois stratégies pour prioriser les objectifs normatifs et les objectifs de l'agent.

1.5.3.3. Stratégies de prise en compte des normes

Pour traiter des normes dans JSAN, il est nécessaire de : (i) spécifier les objectifs d'un agent ; (ii) spécifier différentes stratégies de mouvement d'un agent ; et (iii) fournir des stratégies normatives pour représenter la manière dont les agents traitent les normes. JSAN

dispose déjà de trois stratégies normatives basiques inspirées des types d'agents dans BOID [Broersen et al., 2001] : Rebelle, social, égoïste (cf. Figure 17)

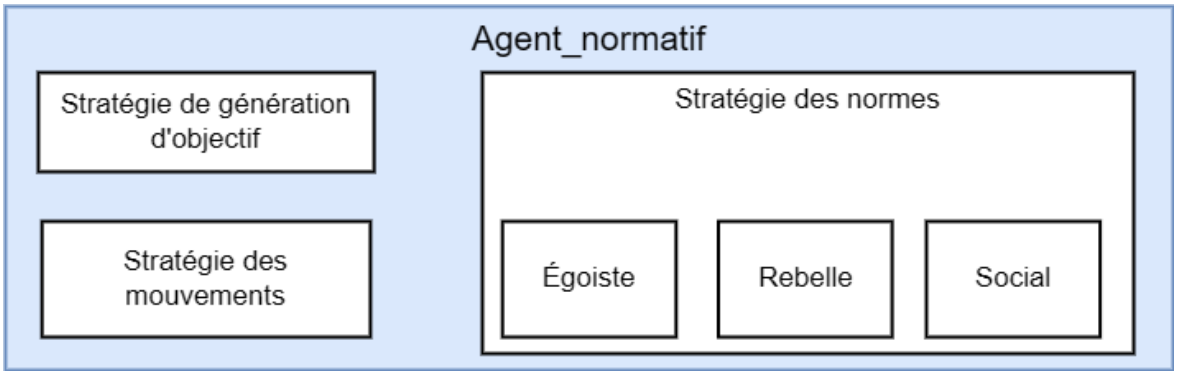


Figure 17: Structures à définir pour créer un agent normatif dans JSAN

Pour prendre en compte les normes dans JSAN nous pouvons donc soit rester sur un typage d'agent classique avec les stratégies Égoïste, Rebelle, ou Social, ou bien créer une nouvelle stratégie de prise en compte des normes en étendant la classe *NormStrategy*.

1.5.4. Synthèse des implémentations des normes en SMA

Dans les revues sur les architectures normatives telles que [Lee et al., 2014] la tendance des implémentations d'agents normatifs les normes par : 1) une acceptation automatique des normes par les agents, et 2) leur prise en compte repose sur des plans prédéfinis. Ces deux hypothèses ont pour conséquences :

1. Une dépendance entre l'implémentation des normes et des plans : Lorsque l'agent rencontre de nouvelles normes non spécifiées lors de l'initialisation du modèle, il n'aura pas de plan pour y faire face, et en subiront automatiquement les potentielles sanctions en raison de leur incapacité à traiter la nouvelle norme;
2. L'absence d'autonomie des agents : le fait de considérer les normes comme des contraintes fortes dépouille les agents de leur autonomie, qui est une caractéristique fondamentale des agents, d'autant plus dans un contexte ouvert où l'on peut avoir plusieurs institutions et plusieurs normes.

Pour pouvoir redonner aux agents leur autonomie décisionnelle, et y intégrer les normes, il est nécessaire de présenter comment les plans peuvent être créés automatiquement par l'agent : c'est la planification automatisée.

1.6. Planification automatisée

Pour pouvoir introduire comment un agent peut de manière autonome engendrer son comportement sans dépendre d'une intervention externe, ni être limité par une bibliothèque de plans ou des normes considérées comme des règles absolues, la planification automatisée est utilisée. Cette section aura pour objectif de déterminer ses différentes déclinaisons et de déterminer leur compatibilité pour l'intégration des normes.

La planification automatisée ou la génération de plans d'action en un ensemble de moyens pour trouver automatiquement ce plan. Par définition, un plan d'actions est une séquence ou un ordre partiel d'actions permettant à un système d'atteindre un objectif souhaité [Hendler et al., 1990]. Dans les architectures délibératives, les plans sont le moteur de l'autonomie des agents pour pouvoir transiter d'une situation initiale S_i vers n'importe quelle situation S_f dans laquelle ses objectifs sont réalisés (cf. Figure 18).

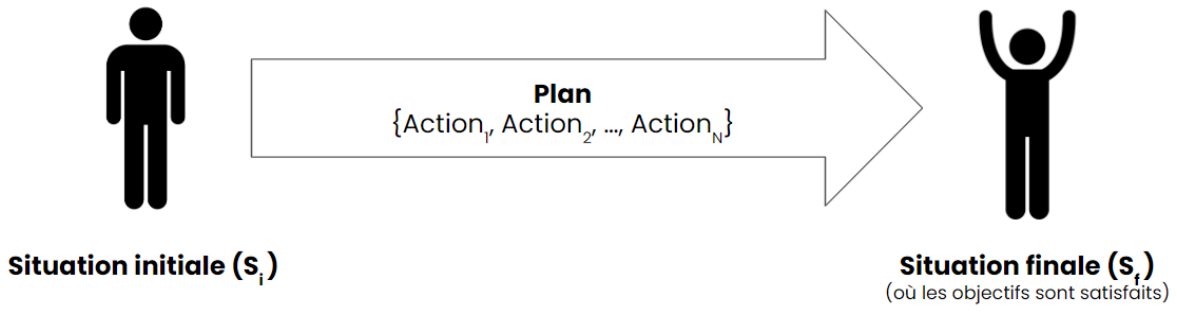


Figure 18 : Illustration globale du principe de la planification

Pour pouvoir mettre en œuvre une planification, il faudra donc spécifier à l'aide d'un langage le problème de planification à résoudre.

Le problème de planification contient : 1) la situation initiale; 2) l'objectif(s) et 3) le répertoire d'actions qui peut être fixe, ou dynamique au fil du temps, et 4) potentiellement l'ensemble des contraintes à respecter pour éviter les incohérences du plan, s'il y en a. Le standard de facto pour formuler un problème de planification est PDDL (Planification Domain Definition Language) [Ghallab et al., 1998] qui lui-même est basé sur STRIPS [Fikes & Nilsson, 1971]. Lorsque le problème de planification est posé, une recherche dans un espace d'états est effectuée pour le résoudre.

Dans l'optique d'introduire comment les normes impactent sur les comportements des agents dans leur architecture, nous présenterons dans ce chapitre i) les standards de facto pour formuler un problème de planification que sont STRIPS et PDDL; puis ii) les différents concepts qui permettent de résoudre ce problème de planification.

1.6.1. STRIPS

STRIPS de [Fikes & Nilsson, 1971] est la toute première référence en termes de planification automatisée classique, il constitue la base du standard de facto : PDDL que nous allons présenter en section 1.6.2.

Définition *(STRIPS) : STRIPS ou STanford Research Institute Problem Solver est un programme de résolution de problème conçu par Fikes & Nilsson qui permet de définir un problème de planification et de trouver un plan, i.e. une séquence d'actions qui le résout automatiquement.*

Nous présentons dans cette section l'ontologie de STRIPS pour décrire un problème de planification, ainsi que les modifications apportées par PDDL, dans l'optique de comprendre comment on peut y intégrer les normes.

1.6.1.1. Définition des prédicats et de ses termes

Pour pouvoir décrire l'état du monde en disant ce qui est vrai (ou faux), le langage utilisé doit pouvoir représenter les objets du système. Ce sont les termes.

Définition *(Terme) : un terme est soit :*

a. *une variable, notée x, y, z,*
b. *une constante notée a, b, c, ou*
c. *un terme fonctionnel noté* $f(t_1, t_2, \ldots, t_n)$ *, composé d'un nom de fonction noté f, g, h, et d'un ensemble de termes* $(t_1, t_2, \ldots, t_n)$.

Il n'existe pas d'autres termes.

Les variables peuvent être liées à des constantes du système pour en décrire une information qui est vraie. Une constante est un symbole qui désigne un objet du système. Le fait de décrire que *x* est un fermier peut être décrit par le prédicat $isFarmer(x)$. Nous pouvons ajouter que *x* a de l'argent avec le prédicat $hasMoney(x)$. Dans ce contexte précis, la variable *x* désigne un individu non défini, que l'on peut spécifier en la substituant à d'autres variables et constantes.

Définition *(Prédicat) : Un prédicat* p *est composé d'un nom, et d'un ensemble de n-termes, noté* $p(t_1, t_2, \ldots, t_n)$ *où p est le nom du prédicat. Un prédicat désigne toujours un fait qui est vrai.*

Les prédicats peuvent être combinés avec plusieurs opérateurs :

- La *conjonction* : qui permet de connecter les prédicats par des "et" logiques;
- La *disjonction* : qui permet de marquer une alternative par des "ou" logiques.

1.6.1.2. Substitution des variables

Définition *(Substitution) : une substitution d'une variable* $x \sim t$ *permet de lier la valeur de* x *à un terme* t.

Une variable peut donc être liée soit à des constantes, soit à d'autres variables qui récursivement peuvent être liées à d'autres variables ou constantes. Pour reprendre l'exemple $hasMoney(x)$, la substitution des variables nous permet de spécifier "qui" désigne *x* dans un

contexte particulier. Par exemple en décrivant $hasMoney(a)$, et $hasMoney(b)$, nous décrivons respectivement les substitutions $x \sim a$ et $x \sim b$.

Lorsqu'une variable n'est pas liée à une constante, la variable est dite *libre*, si elle est liée à une constante, la variable est dite *instanciée*[8]. Avec la notion de prédicats et de substitutions de variables, nous sommes à présent en mesure de décrire ce qu'est une situation.

1.6.1.3. Définition d'une situation

Définition (*Situation****) :*** *Nous définissons une situation* Ş *comme étant une conjonction de prédicats* $\text{Ƥ} = \{p_1, p_2, \ldots, p_n\}$ *où* p_i *est un prédicat dont toutes les variables sont instanciées.*

Ces prédicats décrivent ce qui est vrai dans une situation, et les prédicats qui ne figurent pas dans la situation sont considérés comme faux (hypothèse du monde clos). Par exemple, une situation où un villageois spécifique désigné par une constante *a*, est affamé, et se trouve dans une zone sacrée désignée par la constante *b* peut être décrite par la syntaxe :

$$\text{Ş}_1 = \{villager(a),\ isSacred(b),\ located(a, b),\ starving(a)\}$$

1.6.1.4. Représentation d'action de STRIPS

Définition *(Répertoire d'actions) : Le répertoire d'actions* Ą *est un ensemble potentiellement vide d'actions (avec leurs préconditions et leurs conséquences) tel que* $\text{Ą} = \{a_1, a_n, \ldots, a_n\}$ *où* a_i *est une action.*

Une action modifie l'environnement. Du point de vue de la génération de plans d'action, où seul l'agent modifie l'environnement, on s'intéresse seulement à l'instant avant son exécution et après son exécution.

[8] En anglais, on dit que la variable est *"grounded"*.

Pour qu'une action soit exécutable, ses préconditions doivent toutes être vraies, et après qu'elle soit exécutée, ses post-conditions décrivent les conditions qui deviennent vraies et qui ne sont plus vraies.

Définition *(Action) : Une action est un quadruplet <nom, prec, add-list, delete-list> dont :*

- *le premier est le nom de l'action avec des variables (ses arguments);*
- *le deuxième est l'ensemble des préconditions de l'action;*
- *le troisième est l'ensemble des prédicats qui sont rendus vraies après son exécution;*
- *enfin, le quatrième est l'ensemble des prédicats qui sont rendus faux après son exécution.*

Les préconditions et postconditions des actions sont structurellement représentées par des prédicats ou des négations de prédicats, que nous encapsulons dans la notion d'atôme.

Définition *(Atôme) : Un atôme est un prédicat ou une négation de prédicat.*

Pour les post-conditions des actions, nous pouvons les diviser en deux catégories :

1) La *add-list* : les prédicats qui seront vrais;
2) La *delete-list* : les prédicats qui seront faux.

Formellement, le calcul d'une situation suivante $ş_2$ obtenu par l'exécution d'une action *a* sur une situation précédente notée $ş_1$ se fait en ajoutant à $ş_1$les prédicats de la *add-list* de l'action, et en retirant les prédicats de sa *delete-list*. L'exécution d'une action et le calcul de la situation suivante peuvent s'exprimer avec la formule

$$ş_2 = ş_1 \cup \{add - list\} \setminus \{delete - list\}$$

La représentation d'action de STRIPS admet plusieurs hypothèses qui simplifient la planification :

- Les actions sont déterministes : toutes les post-conditions des actions deviennent toutes vraies après leur exécution;
- Les actions sont instantanées : les actions n'ont pas de notion de durée, mais se basent sur des instants: l'instant avant son exécution où les préconditions sont vraies, et

l'instant après son exécution où les post-conditions sont vraies; de même, les post-conditions des actions deviennent vraies simultanément et instantanément.

Nous pouvons donc résumer la structure d'une action dans STRIPS avec le diagramme de classe UML de la Figure 19.

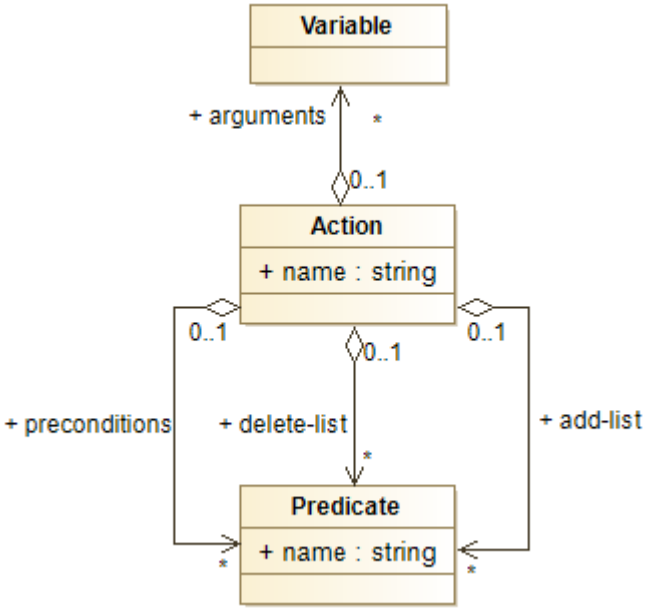


Figure 19: Structure d'une action STRIPS classique

1.6.1.5. Définition d'un objectif

Définition *(Objectif) : un objectif* G *est un ensemble de prédicats que l'on souhaite rendre vrais pour obtenir une situation finale.*

Par exemple, nous pouvons poser comme objectif que l'agent a ait du bois et qu'il ne soit pas en dette, avec l'objectif $G = \{hasWood(a), \neg inDebt(a)\}$.

1.6.1.6. Définition d'un plan

Définition *(plan) : Un plan dans STRIPS est une séquence ordonnée d'actions qui, une fois exécutée, permet de passer d'une situation initiale à une situation qui satisfait tous les atomes de l'objectif.*

D'autres alternatives sont également disponibles pour spécifier un problème de planification, le standard de facto étant PDDL (Plan Domain Definition Language) que nous allons aborder dans les sections suivantes.

1.6.2. PDDL

PDDL ou Plan Domain Definition Language [Ghallab et al., 1998] est un langage formel, basé sur STRIPS qui est considéré comme étant un standard de facto pour déclarer et uniformiser un problème de planification [Jackson et al., 2021, p. 2].

PDDL définit deux fichiers distincts : 1) le problème de planification à résoudre, et 2) le domaine de planification qui contient les prédicats, les fonctions, les types d'objets présents dans le système.

1.6.2.1. Problème de planification

Un exemple de la syntaxe de PDDL, inspirée du langage LISP pour décrire un problème de planification, tel que le monde des blocs, est illustré dans le Code 1.

```
 1 (define (problem blocks-world-problem)
 2   (:domain blocks-world)
 3   (:objects block1 block2 block3)
 4   (:init (clear block1)
 5          (clear block2)
 6          (clear block3)
 7          (on-table block1)
 8          (on-table block2)
 9          (on-table block3))
10   (:goal (and (on block1 block2)
11               (on block2 block3))))
```

Code 1: Illustration d'un problème de planification dans PDDL

La ligne 2 du Code 1 définit le domaine utilisé par le problème de planification notamment le répertoire d'actions.

1.6.2.2. Représentation d'actions de PDDL

Pour déclarer une action classique dans PDDL, il nous faut décrire dans le domaine de planification le schéma des actions comme pour la représentation d'action de STRIPS:

- Le nom de l'action;
- Ses arguments qui sont des variables;
- Ses préconditions qui sont des prédicats ou des négations de prédicats préfixés par le mot *"not"*, et qui représentent des atomes;
- Ses postconditions qui sont également des atomes, les atomes positifs constituant l'équivalent de l'add-list et les atomes négatifs constituant la delete-list.

Pour distinguer les variables des constantes dans PDDL, elles sont préfixées par *'?'*. Par exemple, pour décrire l'action de bouger d'un endroit à un autre, l'action peut être décrite par la Figure 20, avec comme nom "move-self", et la précondition de devoir être à un certain endroit *?m,* différent de l'endroit de destination *?l.* Une fois exécutée l'agent arrive dans *?l* et ne sera plus dans *?m*.

```
1 (:action move-self
2          :parameters (?m ?l - location)
3          :precondition (and (at self ?m) (not (= ?m ?l)))
4          :effect (and (at self ?l) (not (at self ?m))))
```

Figure 20: Description d'une action "bouger" d'une localisation *m* vers *l* en PDDL

Dans ce contexte, *self* est une constante, qui désigne l'agent lui-même. Cette action décrit que l'agent peut bouger d'un endroit ? *m* vers un endroit ? *l*. PDDL permet également de typer les arguments des actions et des prédicats, en précisant à la ligne 2 que *?m* et *?l* doivent être de type *location*.

Plusieurs améliorations ont été apportées dans les versions subséquentes de PDDL. Par exemple, dans sa version 2.1., PDDL propose notamment :

- une syntaxe pour exprimer les quantités avec des opérateurs arithmétiques tels que **"<, >, >=, <=, -, +, /"**, et une affectation directe (avec le mot clé `assign`) ou relative des valeurs numériques (par exemple avec les fonctions incrémenter ou décrémenter);
- une durée aux actions;
- des postconditions d'action conditionnelles;

Pour illustrer quelque une de ces extensions, le Code 2 nous montre la formulation d'une action avec une expression des quantités.

```
1 (define (domain jug-pouring)
2     (:requirements :typing :fluents)
3     (:types jug)
4     (:functions
5         (amount ?j - jug)
6         (capacity ?j - jug))
7     (:action pour
8         :parameters (?jug1 ?jug2 - jug)
9         :precondition (>= (- (capacity ?jug2) (amount ?jug2)) (amount ?jug1))
10        :effect (and (assign (amount ?jug1) 0)
11                     (increase (amount ?jug2) (amount ?jug1)))
12 )
```

Code 2: Illustration des quantités dans PDDL 2.1.

Sur le Code 2, nous avons deux quantités définies : l'eau qu'il y a dans le récipient (*amount*), et sa capacité maximale (*capacity*). Ses préconditions sur la ligne 9 décrivent que pour exécuter l'action, il faut que *jug2* ait assez d'espace pour accueillir la quantité dans *jug1* avec la formule : $capacity\ jug2\ -\ amount\ jug2\ >=\ amount\ jug1$.

Un autre exemple, que nous pouvons citer pour illustrer les différences de STRIPS et de PDDL sont les actions à postconditions conditionnelles, avec des quantificateurs, mais aussi des durées où les postconditions ne sont plus rendues vraies seulement à la fin d'action, mais aussi dès le début de l'action ou à un moment donné au cours de son exécution.

Définition *(Action durative PDDL) :* Soit α une action durative qui débute au temps s et qui se termine au temps e, exécutée sur l'intervalle de temps $[s, e]$. Les composants de α sont :

- Ses préconditions avec les intervalles de temps où elles doivent être vraies;
- Sa durée sous la forme d'un entier positif;
- Ses postconditions avec les intervalles de temps où elles doivent être vraies;

Parmi les opérateurs utilisés pour mettre en œuvre les actions duritaves, PDDL 2.1 possède les opérateurs temporels suivants :

- *at start*: elle décrit que la précondition doit être vraie seulement au début de l'exécution de l'action, et que la postcondition est immédiatement vraie dès le début de l'exécution de l'action;

- *at end* : elle décrit que la précondition doit être vraie seulement à la fin de l'intervalle, et que la post condition devient vraie à la à la fin de la durée de l'action;
- *over all* : elle décrit que la précondition ou la postcondition reste invariante lors de l'intervalle d'exécution $]s \ldots e[$, pour qu'une condition soit vraie durant la totalité de l'action, il faut donc cumuler les prédicats $(at\ start\ p)(over\ all\ p)(at\ end\ p)$
- *duration* : est la durée de l'intervalle $[s, e]$.

Le Code 3 permet d'illustrer un exemple d'action durative dans PDDL.

```
1 (:durative-action load-truck
2     :parameters (?t - truck)
3                 (?l - location)
4                 (?o - cargo)
5                 (?c - crane)
6     :duration (= ?duration 5)
7     :condition (and (at start (at ?t ?l))
8                     (at start (at ?o ?l))
9                     (at start (empty ?c)
10                    (over all (at ?t ?l))
11                    (at end (holding ?c ?o))
12    :effect (and (at end (in ?o ?t))
13                 (at start (holding ?c ?o))
14                 (at start (not (at ?o ?l)))
15                 (at end (not (holding ?c ?o)))
16 )
```

Code 3: Illustration des actions duratives de PDDL 2.1

Dans cet exemple, est décrite une action pour charger un camion (truck), à un certain endroit (location), avec une certaine marchandise (cargo) avec une certaine grue (crane). Elle dure 5 secondes (ligne 5), et sur les lignes 7-11 nous avons les préconditions :

- Au début de l'action, le camion doit obligatoirement être vide à l'endroit prévu ;
- Idem pour la marchandise qui doit être sur place;
- Durant la durée de l'action, le camion doit rester sur le site;
- Et vers la fin de l'action, la grue doit tenir la marchandise, lui laissant la possibilité de ne pas le tenir à certains moments du plan, avant de finir l'action;

D'autres extensions sont également mentionnées dans la littérature telles que : les actions concurrentes, la planification temporelle, et les postconditions continues [Fox & Long, 2003]; les contraintes et les préférences sur les plans [Gérévini & Long, 2005] qui

permettent de servir de critère entre plusieurs plans possibles, ou encore des extensions pour prendre en compte l'espace [Belouaer et al., 2012]. Toutefois, nous ne couvrons pas de manière exhaustive toutes les extensions possibles de PDDL dans le présent travail afin de se focaliser sur la prise en compte des normes.

1.6.3. Algorithme de recherche

Pour résoudre un problème de planification, qu'elle soit formulée à travers STRIPS ou PDDL, l'algorithme construit un espace des possibles, appelé l'espace d'états. L'algorithme maintient la liste des états qui ne sont pas encore parcourus: c'est la liste *open*, et inversement, puisque l'espace de recherche peut être cyclique, l'algorithme maintient la liste des états qui ont déjà été parcourus: c'est la liste *closed*. L'algorithme consiste à parcourir la liste d'états *open*, et pour chacun d'entre eux :

- Trouver l'ensemble des opérateurs applicables;
- Calculer les états successeurs;
- Ajouter les états successeurs dans *open* s'ils ne sont pas déjà dans *closed*;
- S'arrêter si un des états successeurs est un état final;
- Il n'y a pas de plan solution quand le dernier état successeur n'est pas un état final et *open* est vide.

Pour pouvoir décrire en détails cet algorithme dans la Figure 21, nous posons un certain nombre de fonctions abstraites à implémenter, à savoir :

- *getInitialState* : obtenir l'état initial;
- *getOptions* : obtenir la liste des opérateurs possibles dans un état donné
- *apply* : appliquer un opérateur sur un état pour calculer l'état suivant
- *isValid* : tester si l' état est cohérent
- *isFinal* : tester si l'état est terminal
- *evaluateState* : évaluer la distance d'un état à un état final et traiter les états les plus prometteurs en priorité.

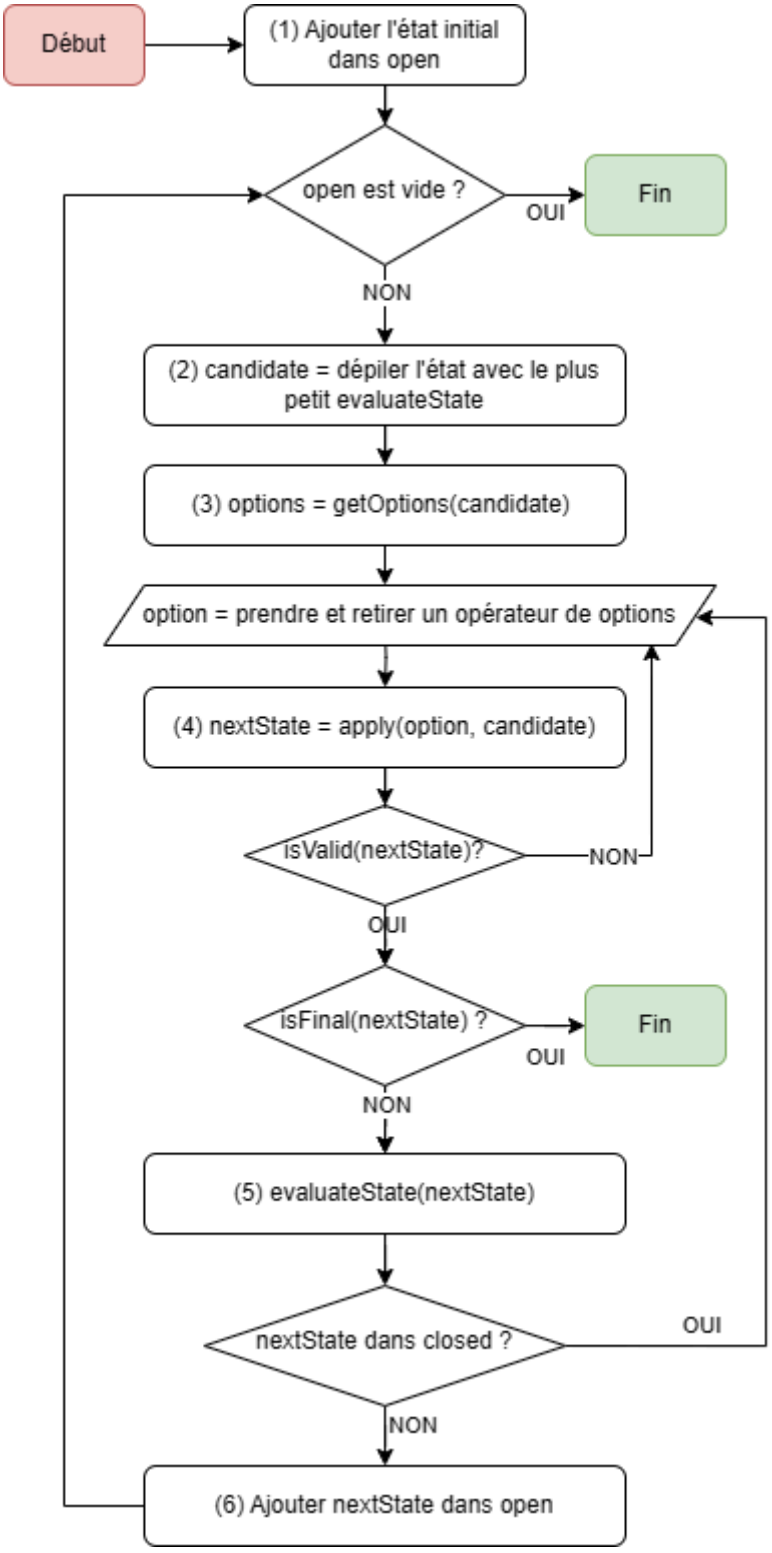


Figure 21: Principe algorithmique d'une recherche dans un espace d'état

Pour décrire plus en détails la Figure 21, nous avons les étapes numérotées suivantes :

1. Au début de l'algorithme, l'état initial est inséré dans la pile des états non explorés, appelée *open*;
2. Si *open* est non-vide: L'état avec la distance heuristique la plus petite est dépilé de *open*, nous appelons cet état *candidate*;
3. L'ensemble des opérateurs applicables à *candidate* est calculé;
4. Pour chaque opérateur applicable, nous calculons l'état suivant. Si celui-ci n'est pas valide, on revient sur l'étape 2, sinon, on vérifie si l'état est un état final. Si oui, la planification s'arrête;
5. Si l'état n'est pas final, son heuristique est calculée;
6. Si le nouvel état n'est pas déjà dans la liste des états traités (*closed*), il est ajouté à *open*, et enfin *candidate* est ajouté, appelé *closed*. Le processus reprend alors depuis l'étape 2.

Parmi les approches illustrées dans [Weld, 1999], nous avons les trois familles d'approches qui définissent différemment ce qu'est un état, un opérateur de la recherche, le critère de terminaison / validité et les options possibles :

1. le chaînage avant, où l'état va être une situation (State) et l'opérateur une action (Operator). L'état initial est la situation initiale (getInitialState), et ses opérateurs possibles seront les actions dont les préconditions sont vraies dans cette situation (getOptions). L'application des opérateurs permet d'obtenir de nouveaux états en ajoutant / enlevant des prédicats selon leurs post-conditions (apply). On s'arrête quand on obtient une situation où l'ensemble des objectifs sont atteints (isFinal), la distance minimum par rapport à un état final est le nombre d'objectifs non atteints (evaluateState);
2. le chaînage arrière, où l'état initial va être un ensemble d'atomes à établir (State), les opérateurs des actions (Operator), l'état initial va être l'objectif à établir (getInitialState), les opérateurs possibles sont les actions dont les postconditions établissent l'un ou l'autre des atomes (getOptions). L'application d'un opérateur ajoute les préconditions à satisfaire (apply), jusqu'à obtenir un ensemble d'atomes satisfait par la situation initiale (isFinal). Un état est cohérent si ses atomes ne sont pas contradictoires (isValid), et sa distance par rapport à un état final est mesurée par le nombre d'atomes non satisfaits par la situation initiale (evaluateState);

3. la planification à ordre partiel (POP), où l'état va être un plan partiel (State), et les opérateurs possibles des modifications du plan (Operator) pour le rendre exécutable et satisfaisant les objectifs (getOptions & isFinal). Un état cohérent est un état où le plan partiel n'est pas contradictoire (isValid), et l'état suivant est calculé en ajoutant les modifications du plan qui permettent de corriger les lacunes du plan (apply). La distance par rapport à un état solution est traduite par le nombre de lacunes du plan qui l'empêche d'être un état final (evaluateState).

Parmi ces trois familles d'approches, la planification en chaînage avant est celle qui est la plus intuitive, et qui ne nécessite aucun nouveau concept (autre que la situation, les objectifs, et les actions qui ont déjà été décrites). Nous proposons dans la suite de cette section, d'expliciter le cas des planificateurs en chaînage arrière et des planificateurs à ordre partiel.

1.6.4. Les planificateurs en chaînage arrière

Le chaînage arrière explore l'ensemble d'un ensemble d'atomes à établir, en partant de l'objectif vers un ensemble d'atomes satisfaie par la situation initiale. Cet ensemble est contradictoire quand on peut déduire par inférence que :

a) un atome p et son contraire $\neg p$ simultanément dans la contrainte; ou
b) des atomes qui sont définis comme incohérents par un ensemble de règles.

Par exemple, l'ensemble d'atome $\{\neg located(a),\ located(a)\}$, est contradictoire. Plus loin, nous pouvons définir une contradiction avec un ensemble de règles décrivant quels atomes ne peuvent pas coexister.

Par exemple, la règle $SI\ emptyHand()\ ALORS\ \neg hold(object)\}$ permet de spécifier qu'il est contradictoire d'avoir à fois la main vide et tenir un objet en même temps. Ainsi, nous pouvons en déduire que l'ensemble $\{emptyHand(),\ hold(a)\}$ est contradictoire. Il est donc nécessaire de fournir l'ensemble des règles de cohérence exhaustives, au risque des plans incohérents : un procédé qui reste difficile pour les modélisateurs.

1.6.5. Les planificateurs à ordre partiel

L'idée des planificateurs à ordre partiel (POP) est de passer d'un plan à ordre partiel à un autre, en modifiant celui-ci pour corriger ses lacunes, jusqu'à ce que le plan permette d'atteindre les objectifs tout en étant exécutable. Pour décortiquer cette idée générale, nous définissons dans cette section :

1) Ce qu'est un état, i.e. un plan partiel;
2) Ce qu'est une lacune dans un plan partiel;
3) Ce qu'est un plan exécutable, et qui satisfait les objectifs;

Définition *(Plan partiel) : Un plan à ordre partiel $\mathcal{P}$ est un 5-uplet tel que* $\mathcal{P} = < \square, \wp, Cc, Ct, F >$ *où :*

- $\square$ *: un ensemble de situations;*
- $\wp$ *: un ensemble de pas;*
- Cc *: un ensemble de contraintes de (non) codénotations;*
- Ct *: un ordre partiel sur* $\square \cup \wp$ *;*
- *F : un ensemble de lacunes du plan*

Le reste de cette section présente les contraintes de (non)codénotation, les contraintes temporelles, les pas, les situations, et la notion de lacune; puis nous présenterons ce qu'est un plan exécutable satisfaisant les objectifs (*isFinal*).

1.6.5.1. Les contraintes de codénotation

Définition *(Contrainte de [non-]codénotation) :* une contrainte de (non) codénotation, notée $x \sim y$ (et $x \nsim y$ respectivement pour une contrainte de non-codénotation) impose que la variable x soit substituée (respectivement ne soit pas substituée) à y. Elle est non-contradictoire si on n'a pas simultanément: 1) $x \sim y$ et $x \nsim y$ et 2) $x \sim a$ et $x \sim b$, avec a et b étant des constantes différentes.

En plus de pouvoir spécifier les substitutions des variables aux objets, les contraintes de non-codénotation permettent également d'interdire certaines substitutions. Par exemple,

avec deux prédicats $located(x)$ et $located(a)$, en posant une contrainte de (non)codénotation $x \nsim a$, il est impossible de les rendre identiques car la codénotation $x \sim a$ n'est pas autorisée. Nous utilisons donc la notion de contrainte de (non)codénotation pour non seulement décrire des substitutions de variables, mais également pour interdire certaines substitutions.

Soient deux atomes p et q, p et q sont unifiables (noté $p \sim q$) s'il existe un ensemble de contraintes de codénotation qui rendent les deux atomes identiques. L'unification permet par exemple d'avoir un prédicat $isSacred(z)$ qui décrit le fait qu'une zone z quelconque est sacrée, et d'en déduire que ce prédicat est identique à $isSacred(a)$, avec la contrainte de codénotation $z \sim a$.

1.6.5.2. Définition d'un pas

Définition *(pas) :* Un pas est une instance d'action à laquelle on associe une situation d'entrée dans laquelle il faudra assurer que ses préconditions sont vérifiées, et une situation de sortie dans laquelle ses postconditions sont établies.

Chaque pas rend vrais les prédicats de sa *add-list* dans la situation qui le succède, et inversement, il rend faux les prédicats de sa *delete-list* dans la situation qui le succède. Nous pouvons de ce fait formaliser quand est ce qu'un atome p est vrai dans une situation *s*.

Définition *(critère de la vérité) :* un atome p est vrai dans une situation s si et seulement si $\exists q \in s$ tel que $p \sim q$.

Pour qu'un atome soit vrai dans une situation, il faut qu'un pas du plan qui précède cette situation l'engendre au préalable par ses postconditions (add-list) ou qu'il soit vrai dans la situation initiale et ne soit pas potentiellement détruit par un pas (delete-list). On dit qu'un pas λ *asserte* un atome p s'il existe un atome q dans ses postconditions tel que $p \sim q$. Inversement, on dit qu'un pas λ *détruit* un atome p s'il existe un atome q dans ses postconditions tel que $\neg p \sim q$.

Nous utiliserons le critère de la vérité nécessaire de [Chapman, 1987] pour désigner le fait qu'un pas *rend nécessairement vrai* un atome, c'est-à-dire que l'atome est vrai dans toutes les complétions possibles du plan.

Définition *(critère de la vérité nécessaire) :* pour qu'un atome p soit nécessairement vrai dans une situation s, noté $\square(p, s)$ il suffit que :

1) L'atome p soit asserté dans une situation e qui précède ou est identique à s;
2) Pour l'ensemble D des pas qui détruisent p :
    a) Ils se trouvent tous après s ou avant le pas ξ qui asserte p;
    b) Sinon, si un pas d qui détruit p existe tel que $\xi \prec d$ et $d \prec s$, p est nécessairement vrai dans s s'il existe un autre pas r qui réasserte p tel que $d \prec r$ et $r \prec s$ et $\exists q \in postconditions(r)$ tel que $q \sim p$.

1.6.5.3. Les contraintes temporelles

Définition *(Contrainte temporelle) : une contrainte temporelle permet de désigner quel pas est avant l'autre, quelle situation est avant l'autre, et quelle situation est avant quel pas.*

Par exemple, une contrainte temporelle notée $s_i \prec s_j$ décrit que le pas (ou situation) s_i est devant le pas (ou situation) s_j. Elle doit définir un ordre partiel (pas de boucles), sinon, elle est considérée comme contradictoire. Un exemple de contrainte temporelle contradictoire serait la contrainte temporelle $Ct = \{ s_i \prec s_j, s_j \prec s_k, s_k \prec s_i \}$ (puisqu'elle forme une boucle).

Pour simplifier les contraintes temporelles, nous nous basons uniquement sur la relation de précédence notée $\prec$, nous ne prenons pas en compte les relations de succession (qui serait notée $s_i \succ s_j$ pour dire que s_j se situe après s_i).

1.6.5.4. Les lacunes

Définition *(lacune) : Une lacune désigne ce qui empêche un plan à ordre partiel d'être exécutable, il s'agit de l'ensemble des préconditions des pas du plan qui ne sont pas encore nécessairement vraies dans la situation qui les précède.*

Une lacune est soit une condition ouverte, soit une menace :

- *Une condition ouverte: un des atomes de la précondition d'un pas P n'est assertée par aucun autre pas qui précède possiblement la situation d'entrée de P;*
- *Une menace : un des atomes de la précondition d'un pas P est bien assertée par un pas E du plan dans sa situation d'entrée, mais il n'est plus nécessairement vrai à cause de la delete-list d'un autre pas, qui est possiblement entre E et P.*

Pour pouvoir solutionner ces lacunes, des modifications au plan doivent être appliquées : que ce soit de nouvelles situations, de nouveaux pas, de nouvelles contraintes de (non) codénotation et/ou des contraintes temporelles.

Formellement nous noterons :

- s^{a}_{before}: la situation qui précède le pas a;
- s^{a}_{after}: le situation après l'exécution du pas a;

Pour résoudre une condition ouverte $p \in preconditions(a)$ que l'on souhaite rendre nécessairement vraie dans la situation s^{a}_{before} qui précède le pas a, nous pouvons utiliser les opérateurs suivants:

1) *Par Création* : ajouter un nouveau pas ξ dans le plan qui asserte la condition ouverte p, et le placer devant s^{a}_{before}, avec les contraintes de codénotation adéquates pour que $\exists q \in postconditions(\xi) \mid q \sim p$.

On ajoute donc les modifications suivantes pour l'opérateur de création :

$$\square = \square \cup (s^{\xi}_{before}) \cup (s^{\xi}_{after})$$

$$\wp = \wp \cup \xi$$

$$Cc = Cc \cup (q \sim p)$$

$$Ct = Ct \cup (s^{\xi}_{before} \prec \xi) \cup (\xi \prec s^{\xi}_{after}) \cup (s^{\xi}_{after} \prec s^{\alpha}_{before})$$

2) *Par Promotion :* mettre un pas existant ξ avant la situation s^{a}_{before} où p n'est pas nécessairement vraie, avec les contraintes de codénotation (si nécessaires) pour faire en sorte qu'il asserte p à travers une de ses postconditions notée *q*. On ajoute donc la modification suivante :

$$Ct = Ct \cup (s^{\xi}_{after} \prec s^{a}_{before}); Cc = Cc \cup (q \sim p)$$

La menace d'un pas d qui détruit, *i.e.* qui détruit possiblement la précondition p d'un autre pas a peut être résolu :

- *Par Demotion* : on s'assure de mettre le pas destructeur d après a en ajoutant la contrainte temporelle $a \prec d$, ou plus précisément $s^{a}_{after} \prec s^{d}_{before}$;
- *Par un rétablisseur :* on ajoute un nouveau pas ou on prend un pas existant r qui re-asserte la précondition *p* de l'action *a* après qu'il soit détruit avec les contraintes temporelles $\{d \prec r, r \prec a\}$
- *Par Non-codenotation*: ajouter une contrainte de non-codénotation de sorte à ce que $\nexists q \in postconditions(d)$ tel que $\neg p \sim q$.

1.6.5.5. Exécutabilité d'un plan

Définition *(Plan exécutable) : Un plan est exécutable lorsque, pour chaque pas du plan, chacune de ses préconditions est nécessairement vraie dans la situation qui le précède.*

Puisque dans un plan exécutable, chaque précondition de chaque pas est établie correctement, un plan exécutable est tout simplement un plan sans aucune lacune, et qui n'a pas besoin d'être un plan complet.

Définition *(Plan complet) : Un plan est complet si :*

1) *ses contraintes temporelles définissent un ordre total sur ses pas et ses situations;*
2) *Ses contraintes de codénotation font codénoter chaque variable à une constante.*

En se basant sur le vocabulaire établi nous pouvons donc dresser les composants du POP :

- *State* : un plan à ordre partiel;
- *Operator* : désigne une modification du plan par ajout d'un ensemble de situations, de pas, de contraintes temporelles, et/ou de contraintes de (non) codénotation;
- *getInitialState* : retourne un plan partiel initial que nous allons décrire dans cette section;
- *getOptions* : retourne l'ensemble des résolutions de toutes les lacunes du plan à ordre partiel;
- *apply* : ajoute les modifications au plan précédent pour obtenir le plan suivant;
- *isValid* : un plan est valide lorsque ses contraintes temporelles et ses contraintes de (non) codénotation sont toutes les deux non-contradictoires.
- *isFinal* : retourne *vrai* si l'état est un plan est exécutable;
- *evaluateState* : calcule le nombre de lacunes non-résolues du plan;
- *evaluateOperator* : retourne 1 par défaut, mais peut être personnalisé si l'on souhaite dresser une priorité entre la distance de l'état par rapport à un état final, et la distance du parcours dans l'espace de recherche.

L'état (=plan partiel) dans le POP spécifie un ensemble de plans complets. Pour avancer dans l'espace de recherche, on doit détecter des lacunes dans les états, et les amender, la littérature sur le POP distingue généralement deux types de lacunes :

- Les conditions ouvertes (OC), qui sont des préconditions d'un pas du plan qui ne sont pas nécessairement vraies dans la situation qui les précède;
- Les menaces (T), qui sont des conflits entre deux pas non ordonnés entre eux du plan, où un pas peut avoir comme post-condition la négation de la précondition d'un autre;

1.6.5.6. Satisfaction des objectifs

Dans le POP, un plan est un état final s'il est exécutable et satisfait les objectifs. Pour satisfaire les objectifs, l'état initial (getInitialState) est composé d'un 5-uplet initial $< \square, \wp, Cc, Ct, F >$ avec :

1. $\square$: $\{s_i , s_f\}$ où S_i désigne la situation initiale, et S_f désigne la situation finale où les objectifs sont atteints
2. $\wp$: $\{P_i , P_f\}$ tel que P_i est un pas fictif qui asserte la situation initiale, et P_f est un pas fictif qui a pour préconditions l'objectif désiré;
3. Cc : $\emptyset$
4. Ct : $\{S_i \prec S_f, P_i \prec S_i, S_f \prec P_f\}$
5. F : les lacunes du plan, notamment les conditions ouvertes dans S_f

Ainsi, il est garanti que les objectifs sont satisfaits dans la situation finale : puisque les préconditions du pas fictif final P_f (l'objectif désiré) ne sont pas nécessairement vraies dans la situation S_f des lacunes sont présentes dans le plan pour permettre au planificateur de les rendre nécessairement vrais.

La condition de terminaison (isFinal) de la planification à ordre partiel se résume donc à chercher un plan exécutable. Cette représentation est reprise sous une autre forme plus simplifiée dans la Figure 22.

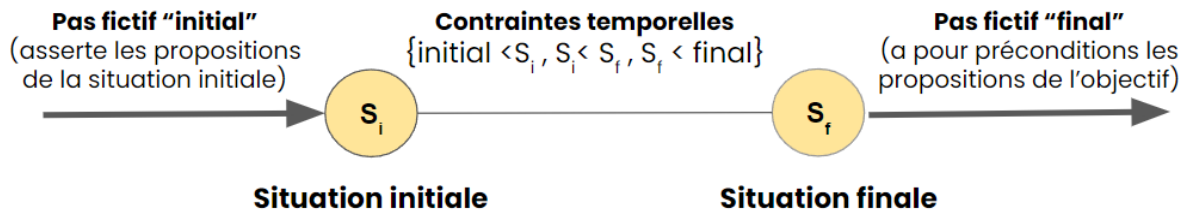


Figure 22: Représentation simplifiée de l'état initial du POP

D'autres notions telles que celle du lien causal, introduite par [McAllester & Rosenblatt, 1991] sont également observées dans la littérature sous le nom de POCL (*Partial Order Causal Link*) pour garder une trace de *"pourquoi"* (pour rendre nécessairement vraie quelle précondition d'action ?) on a ajouté tel pas.

1.6.6. Planification par réseau hiérarchisé de tâches

Une des alternatives aux planificateurs classiques, sont les planificateurs HTN (Hierarchical Task Network) [Erol et al, 1994], que nous pouvons traduire littéralement comme étant "un réseau hiérarchisé de tâches". Elle se focalise sur la résolution de tâches abstraites en les décomposant en tâches plus concrètes grâce à un ensemble de méthodes de réduction, jusqu'à ce qu'obtenir des tâches purement primitives, *i.e.* des actions. Dans HTN un problème de planification $P = (d, I, D)$ est composé de :

1) d : le réseau de tâches initial qui doit être résolu;
2) I : est la situation initiale telle qu'elle est décrite dans STRIPS;
3) D : un domaine de planification, tel que $D = (A, M)$ où A est un ensemble fini d'actions à la STRIPS, et M un ensemble fini de méthodes de réduction fournie par l'utilisateur;

Pour offrir une vue globale des HTN dans le cadre limité de ce travail, nous allons associer le vocabulaire utilisé pour énumérer les planificateurs basés sur STRIPS, ainsi nous aurons :

- *State* : un réseau de tâches $H = (T, C)$ qui est un ensemble de tâches T ordonnées par un ensemble de contraintes temporelles C ;
- *Operator* : une méthode $m = (s, t, H')$ dans M permettant de décomposer une tâche composée en tâches plus primitives pour obtenir un nouveau réseau de tâches;
- *getInitialState* : un réseau de tâches initial d;
- *getOptions* : retourne l'ensemble des méthodes m applicables, i.e. les méthodes de réduction dont les préconditions *(pre(m))* sont satisfaites dans l'état actuel.
- *apply :* l'état suivant se fait par réduction, i.e. en transformant une tâche t au sein d'un réseau de tâches d, à l'aide d'une méthode m.

- *isValid : retourne toujours vrai;*
- *isFinal* : un état est final si le réseau de tâches est composé uniquement de tâches primitives;
- *evaluateState* : à définir selon l'utilisateur, il peut être intuitivement assimilé au nombre de tâches composées restantes dans le plan.
- *evaluateOperator* : arbitrairement égal à 1 si toutes les décompositions se valent.

Une méthode *m* est définie dans [Ghallab et al., 2004; Meneguzzi & de Silva; 2015] par un tuple $m = (s, t, H')$ où *s* est la précondition notée *pre(m)* qui spécifie la condition doit rester vraie pour qu'une tâche *t* (notée *task(m)*) puisse être décomposée en $H' = (T', C')$ qui est noté *network(m)*.

Un réseau de tâches HTN, est une paire $H = (T, C)$ où *T* est un ensemble fini de tâches, et *C* est l'ensemble des contraintes temporelles sur les tâches *T* pour obtenir un ordre total, semblable aux contraintes temporelles dans la planification à ordre partiel. Une tâche peut être primitive ou composée (également dite "abstraite"). L'algorithme de résolution d'un problème de planification HTN se déroule selon les étapes suivantes :

1) Une méthode de réduction *m* applicable est choisie dans *M*;
2) Elle est appliquée à une tâche composée dans *d* pour obtenir un réseau de tâches plus primitives *d'*;
3) Ensuite, une méthode de réduction est appliquée à une tâche composée de *d'* jusqu'à ce qu'on obtienne un réseau de tâches purement primitives;
4) S'il n'existe aucune méthode de réduction pour une tâche composée, le planificateur retourne à une précédente tâche composée et essaye une autre méthode de réduction.

Un des atouts de HTNs qui les rend plus pratiques que des planificateurs STRIPS est le fait d'avoir un espace de recherche généralement plus petit, du fait qu'il cherche toujours à chaque cycle à avoir un réseau de tâches purement primitives, plutôt que de parcourir l'ensemble des objectifs possibles à chaque itération du cycle. Les HTN sont particulièrement adaptés pour des cas où les tâches sont organisées hiérarchiquement, où la décomposition masque plusieurs détails complexes. Le désavantage est que les conflits entre les différentes actions complexes et les actions primitives peuvent également être masqués.

1.6.7. Planification par chaînes de Markov

Les chaînes de Markov (ou processus stochastique) se basent sur le point de vue que les perceptions de l'agent et / ou les conséquences des actions sont incertaines. Contrairement aux planificateurs déterministes, les approches stochastiques assument qu'une action peut avoir plusieurs résultats, avec pour chaque résultat une probabilité d'occurrence associée.

Un problème de planification, appelé un MDP (Markovian Decision Problem) selon [Shoham & Leyton-Brown, 2008] est décrit comme étant un tuple $\Sigma = < SS, A, Pr, R >$ où :

- *SS* est un ensemble fini d'états;
- *A* est un ensemble fini d'actions;
- *Pr* est un système d'états-transitions noté $Pr_a(s|s')$ qui définit la probabilité de transitionner d'un état $s \in SS$ à un état suivant $s' \in SS$ en effectuant l'action a;
- *R* est une fonction d'utilité symbolisant la récompense qui assigne une valeur $r(s_i, a_j)$ aux choix d'actions a_j dans l'état s_i qui peut être utilisé pour décrire le meilleur état possible.

Pour rester consistant dans ce manuscrit, nous pouvons associer les processus stochastiques au vocabulaire de comparaison comme suit :

- *State* : une situation STRIPS classique ;
- *Operator* : est l'application d'une action sur un état ;

Pour mettre en perspective une architecture d'agent POMDP, la Figure 23 présente son cycle de perception-action. L'estimateur perçoit l'état du monde en incluant l'aspect stochastique des perceptions. La politique[9] d'un POMDP est la composante chargée de transformer une série d'observations en actions. Une revue plus détaillée des algorithmes utilisés par les politiques en POMDP est présentée dans [Braziunas, 2003]

[9] En anglais "policy"

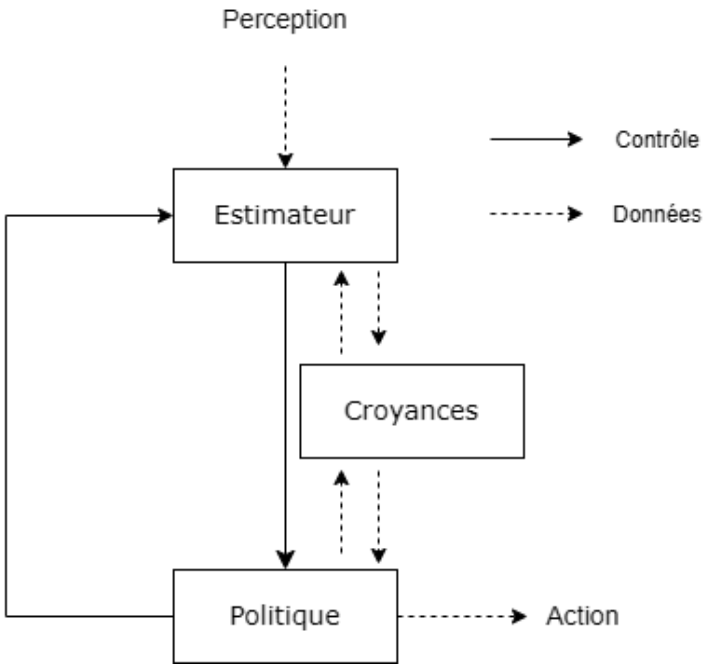


Figure 23: Cycle de perception-action d'un agent POMDP, adapté de [Schut et al., 2012].

Lorsque ce processus décisionnel basé sur les chaînes de Markov, inclut des actions et des perceptions stochastiques, elles deviennent des processus stochastiques partiellement observables (abrégé en POMDP[10]). Le lien entre les composants d'un POMDP et l'architecture BDI est avancé par [Schut et al., 2002]. Soient :

- Un agent décrit par un MDP est un tuple $< SS_{MKV}, A, Pr, R >$ avec un ensemble d'états *SS*, un ensemble d'actions *A*, une fonction de transition *Pr*, et une fonction de récompense *R*;
- Un agent décrit par un POMDP, ajoute un ensemble d'observations O avec une probabilité d'émission □, ce qui donne le tuple $< SS_{MKV}, A, Pr, R, O, \Omega >$.
- un agent BDI est un tuple $< SS_{BDI}, A_{BDI}, Bel, Des, Int >$ où SS_{BDI} est l'ensemble des états de l'agent, A_{BDI} est le répertoire d'actions de l'agent, *Bel* est l'ensemble des croyances décrites par un sous-ensemble de SS_{BDI}, *Des* l'ensemble des désirs, et *Int* l'ensemble des intentions;

Nous avons les correspondances suivantes entre l'architecture BDI et les POMDP :

[10] POMDP : Partially Observable Markov Decision Processes, également traduits en tant que "Chaîne de Markov partiellement observables.

- L'ensemble des états observables par l'agent dans POMDP noté SS_{MKV} est associé à l'ensemble des croyances de l'agent BDI, i.e. $SS_{MKV} \equiv Bel$;
- Les fonctions de récompenses R correspondent aux désirs *Des* de l'agent, puisque maximiser les récompenses à travers ses actions s'aligne aux désirs de l'agent. Les deux peuvent être utilisées pour articuler une préférence sur les états possibles;
- L'ensemble des actions dans un POMDP est associé aux actions de l'agent qui lui sont externes, i.e. $A_{MKV} \equiv A_{BDI}$;
- La fonction de transition $Pr_a(s|s')$ de POMDP est assimilable à la fonction de transition de l'agent BDI à travers l'application des actions. De même, la perception de l'environnement qui met à jour les croyances dans l'architecture BDI, est semblable à l'estimateur d'état de POMDP;
- L'ensemble des observations *O* est associé à *Bel* (aux croyances de l'agent);
- Une combinaison des actions et des récompenses peut être mise en correspondance avec les intentions de l'architecture BDI, puisqu'une intention est tout simplement l'association d'un désir (qui correspond aux récompenses) et un plan d'actions (qui correspond à un ensemble d'actions dans POMDP

Les chaînes de Markov et les POMDP proposent des approches de planification pour un monde non-déterministe, autant dans les postconditions des actions que dans les perceptions du monde si le phénomène étudié le requiert, y compris pour les normes.

Toutefois, les POMDP demandent à quantifier leur probabilité d'occurrence pour fonctionner, et le résultat peut contenir plusieurs instances de la même action puisque l'action peut échouer, et ne produit pas de postconditions.

La question est de considérer la possibilité des échecs des règles de perception-action, comme un concept exploitable pour la prise en compte des normes, mais celle-ci réside au-delà du présent travail, tout comme la prise en compte des normes comme étant des concepts partiellement observables; nous nous limitons à un un monde déterministe pour plus de simplicité et d'explicabilité.

1.6.8. Synthèse sur la planification automatisée

Les exemples illustrés précédemment démontrent de la pléthore de techniques à disposition pour formuler un problème de planification sous la forme d'un problème de recherche dans un espace d'états et le résoudre. L'approche la plus connue de la planification automatisée classique est le planificateur STRIPS de [Fikes & Nilsson, 1971] qui constitue la base de plusieurs planificateurs plus modernes et plus performants.

Quant à la planification HTN, on la retrouve surtout dans les implémentations de l'architecture BDI en raison de leur similarité [De Silva, 2009] où le principe est de décomposer des tâches plus complexes en tâches atomiques plus simples jusqu'à obtenir un plan entièrement composé d'actions atomiques. Le modélisateur doit donc fournir l'ensemble des méthodes pour décomposer les tâches complexes en tâches atomiques.

La planification à ordre partiel (POP) présente plus de flexibilité de par sa stratégie de moindre engagement : l'ordre dans lequel on enchaîne les actions n'a pas besoin de définir un ordre total, tout comme les liaisons des variables, c'est-à-dire les liens entre variables et objets peuvent être partiels, et des variables peuvent ne pas codénoter sur des constantes.

Par rapport à la prise en compte des normes, le parcours d'un espace de recherche dans le cadre d'une planification automatisée doit prendre en compte les critères suivants :

- Quelles normes sont encore applicables dans quelles situations, en cherchant parmi les situations précédentes les autres normes qui ont été activées et dont la condition de désactivation n'a pas encore été satisfaite ;
- Le parcours d'espace de recherche doit pouvoir reconnaître quand est-ce qu'une norme applicable a été satisfaite ou non : 1) pour le cas d'une obligation, l'agent ne doit pas chercher à satisfaire une obligation déjà satisfaite, 2) pour le cas d'une interdiction, l'agent ne doit pas chercher à amender son plan pour satisfaire une interdiction quand l'objet de l'interdiction n'est pas présent; 3) pour le cas d'une permission, l'agent ne doit pas chercher à amender son plan pour modifier ses actions alors que ces dernières sont toutes permises;

La planification automatisée à travers le parcours d'espace d'états doit donc se donner les moyens de représenter les normes (y compris les institutions et les organisations), de vérifier leur applicabilité, mais aussi leur application, et des opérateurs pour pouvoir les appliquer correctement.

1.7. Planification automatisée normative

1.7.1. Panagiotidi, Vazquez-Salceda et Dignum

Dans [Panagiotidi & Vazquez-Salceda, 2011; Panagiotidi et al., 2013], un planificateur normatif est proposé en extension de la planification classique de STRIPS pour définir 1) les stratégies de prise en compte des normes dans la planification automatisée, 2) les stratégies de violation de normes.

1.7.1.1. Représentation des normes

Pour représenter les normes, l'approche en question considère les modalités déontiques d'obligation et d'interdiction, et exprime une norme *N* avec la formule:

$$N = \{id, type, C_{act}, C_{deact}, C_{maint}, C_{repair}\}$$

où :

- *id* est un identifiant
- *type* est la modalité déontique qui est soit une obligation soit une interdiction,
- C_{act} représente les conditions d'applicabilité de la norme,
- C_{deact} représente les conditions qui mettent fin à l'applicabilité de la norme,
- C_{maint} représente les conditions de maintenance de la norme, qui sont utilisées pour déterminer s'il y a violation de la norme ou non;
- C_{repair} sont les conditions qu'il faut appliquer pour réparer une violation de normes.

Une norme est active(=applicable) si : a) ses conditions d'activation sont satisfaites dans une situation et si ses conditions de désactivation sont insatisfaites, ou b) si elle a déjà été active dans la situation précédente, et que les conditions de désactivation de la norme sont insatisfaites. L'activation d'une norme se calcule donc récursivement pour évaluer si elle a été activée ou non dans les situations précédentes, jusqu'à remonter à la situation initiale.

1.7.1.2. Représentation d'un problème de planification

[Panagiotidi et al., 2013] définit un problème de planification avec trois éléments : 1) une situation initiale, un objectif et un modèle normatif. Puisqu'une situation et un objectif sont respectivement composées d'atomes comme STRIPS, il suffit de définir ce qu'est un modèle normatif.

Définition *(Modèle normatif).* Un modèle normatif est un tuple $M =< F, A, N >$ où F est l'ensemble des atomes possibles, A est un ensemble d'actions, et N est un ensemble de normes.

Cette définition étend donc le problème avec un ensemble de normes sous la forme d'interdictions ou d'obligations, qui sont rattachées au problème de planification à l'aide du composant qu' est le modèle normatif. Les normes sont ainsi utilisées comme des restrictions sur la trajectoire du plan calculée par le planificateur.

1.7.1.3. Technique de planification utilisée

Le planificateur utilisé est basé sur 2APL [Dastani, 2008] comme langage de programmation pour créer des agents BDI pourvus de croyances, d'objectifs, d'actions, de plans prédéfinis, d'événements, et de trois types de règles (pour générer les plans accomplissant les objectifs, pour gérer les évènements internes/externes & les messages, et pour gérer la réparation de plans échoués). Un agent 2APL typique dispose d'un processus délibératif en 5 étapes :

1) Les règles sont vérifiées, et tout nouveau plan produit est ajouté à la liste des plans actifs;
2) La première action de tous les plans est exécutée, après exécution, les objectifs de l'agent dans ses croyances sont vérifiés, s'ils sont accomplies, ils sont retirés, ainsi que les plans qu'ils ont engendré;
3) Les événements externes sont traités, pour une mise à jour des croyances;
4) Gestion des plans échoués;
5) Traitement des messages externes.

Toutefois, les auteurs troquent la bibliothèque de plans pré-compilés pour un générateur de plans d'actions appelé *Metric-FF. Metric-FF* est un planificateur en chaînage avant qui peut résoudre des problèmes exprimés en PDDL 2.1 avec une recherche en largeur dans l'espace des situations avec la stratégie heuristique EHC (*Enforced hill-climbing*) qui choisit le meilleur état successeur à chaque étape de la recherche.

Pour pouvoir vérifier l'applicabilité des normes, le planificateur a besoin non seulement des atomes de la situation actuelle, mais également du statut des normes dans les états précédents. Par exemple, pour déterminer si une norme est désactivée, il faut savoir si elle a été activée au préalable, en plus de déterminer si ces conditions de désactivation sont vraies dans la situation actuelle.

La planification s'arrête quand les objectifs sont atteints et toutes les obligations qui dérivent de la violation d'autres normes, i.e. l'obligation de les réparer sont toutes résolues. Par rapport au vocabulaire que l'on s'est fixé dans la section 1.6.3. Algorithme de recherche, le planificateur Metric-FF de [Panagiotidi et al., 2013] peut s'exprimer similairement à une planification en chaînage avant avec :

- *State*: un ensemble de propositions atomiques (= atomes), et donc une situation;
- *Operator* : une action avec des préconditions et des postconditions;
- *getInitialState*: la situation initiale décrite dans le problème de planification normative;
- *getOptions* : dans toute situation, l'agent peut choisir parmi les actions applicables à une situation donnée, ou bien accomplir une obligation, ou bien réparer une violation d'une norme;

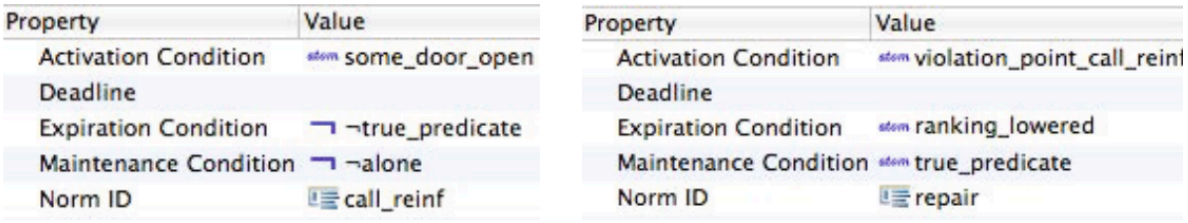
- *isFinal* : tous les objectifs sont accomplis, toutes les normes ont été prises en compte, les réparations des violations d'autres normes sont toutes accomplies;
- *apply* : pour calculer la situation suivante, deux situations intermédiaires sont introduites pour pouvoir 1) d'abord, appliquer l'action sans mettre à jour le statut des normes, 2) effectuer les mises à jour sur le statut des normes (activation, violation, activation d'une norme réparatrice) en vérifiant notamment le statut des normes dans les situations précédentes;
- *evaluateState* : elle se mesure en fonction de la prise en compte des normes, de la violation des normes et des réparations de violations de normes en cours;
- *evaluateOperator* : le coût d'une action qui est à définir par le modélisateur;

1.7.1.4. Stratégies de prise en compte des normes

L'idée globale des auteurs est d'avoir des violations de normes qui seront réparées. Il y a violation de norme lorsqu'une norme est active, et les conditions pour la désactiver ne sont pas remplies, et que les conditions à maintenir ne sont pas satisfaites.

Une violation de norme entraîne l'activation d'une nouvelle norme "réparatrice" pour réparer la violation, c'est-à-dire que la norme réparatrice a pour condition d'activation la violation de la norme.

Pour illustrer les normes et les normes réparatrices, les auteurs prennent comme exemple une simulation d'incendie avec des pompiers qui doivent sauver des personnes dans un immeuble [Panagiotidi et al., 2013]. Ils ont pour interdiction de ne pas rester seuls dans un immeuble en feu si une porte est ouverte. Quand celle-ci est violée, il est obligatoire de diminuer le rang de l'agent en tant que pompier pour le sanctionner. Cette norme ci permet de "réparer" la violation de l'interdiction. Ces exemples sont repris sur la Figure 24.



(a) Une interdiction nommée call_reinf : Si une porte est ouverte dans un immeuble en feu, il est interdit de rester seul.

(b) Une obligation (réparatrice) : Si la norme "call_reinf" est violée, il est obligatoire de diminuer le rang de l'agent.

Figure 24: Un exemple de norme et de norme réparatrice [Panagiotidi et al., 2013]

Pour choisir une option, elle se base sur le coût de chaque action et le coût de chaque réparation en cas de violation d'une norme. Ceci permet à l'agent de choisir entre deux options :

a) Un plan qui adopte la norme;
b) Un plan qui viole une norme, pour réparer la violation par la suite.

La principale contribution des auteurs permet aux normes d'influencer le comportement des agents de manière flexible et pratique en construisant un plan. Ceci permet au modélisateur de spécifier seulement ce qui doit être obligatoire ou interdit, sans avoir à décrire "comment" les agents doivent s'y prendre.

En fonction du degré d'importance que l'agent accorde aux normes et à ses objectifs personnels, le comportement produit peut varier. Plutôt que de trancher à l'aide d'un type d'agent statique, et à toujours choisir de violer ou de respecter les normes comme le cas de BOID, les agents sont capables de percevoir les conséquences d'une violation ou d'une prise en compte des normes.

Le désavantage principal, est que le planificateur marche très bien uniquement dans des environnements où les conséquences de l'accomplissement d'une norme sont connues ou peuvent être estimées, ce qui n'est pas toujours faisable. De plus, cette estimation ne prend pas en compte le fait que les sanctions soient non déterministes : elles peuvent ne pas se produire.

Il s'agit donc d'un planificateur surtout adapté pour des thématiques à données quantifiables. Dans [Panagiotidi et al., 2013], un planificateur TLPLAN [Canan & Birtürk, 2006] est utilisé comme base pour la planification, et permet de retourner des plans prenant en compte les normes, mais qui ne peut pas minimiser les violations de normes dans le cas où aucun plan prenant en compte les normes n'est possible, comme l'indique [Krzisch & Meneguzzi, 2017].

1.7.2. Meneguzzi et al.

Le générateur de plans d'actions de [Krzisch & Meneguzzi, 2017] spécifie avec PDDL le problème de planification et son domaine (les prédicats, les actions, et les objets possibles de l'environnement).

1.7.2.1. Représentation des normes

On distingue deux types de normes dans les travaux de [Krzisch & Meneguzzi, 2017]. Le premier type de norme, dite norme "conditionnelle", est un tuple $n = < \mu, \chi, \rho, C >$, où :

- $\mu \in \{ obligation, prohibition \}$ est la modalité déontique;
- χ est la condition d'applicabilité de la norme, i.e. un ensemble de prédicats;
- ρ est une action qui représente l'objectif de la norme;
- C est la pénalité (coût) de la violation de la norme.

Pour illustrer cette structure, un exemple de norme serait la formulation suivante :

$$cn =< obligation, located(agent, Madagascar), driveRight(agent), 20 >$$

Elle décrit que si l'agent se trouve à Madagascar, il doit rouler sur le côté droit de la route. Une norme conditionnelle $n = < \mu, \chi, \rho, C >$ est violée dans un état *s* si et seulement si s satisfait χ et que :

a) Une action ρ est exécutée dans *s* et μ désigne une interdiction;

b) Une action ρ n'est pas exécutée dans *s* et μ désigne une obligation.

Le deuxième type de norme est défini à l'aide d'une formule exprimée avec une Logique Temporelle Linéaire ou LTL (Linear Temporal Logic) qui est décrite par les opérateurs modaux de PDDL3 [Gerevini & Long, 2005]. Une norme, dite une norme "LTL" est exprimée par l'un des opérateurs modaux suivants qui représentent tous des obligations :

- $(at\ end\ \phi)$: ϕ doit être vrai dans l'état final;
- $(always\ \phi)$: ϕ doit être vrai dans tous les états du plan;
- $(sometime\ \phi)$: ϕ doit être vrai dans au moins un état du plan;
- $(at\ -\ most\ -\ once\ \phi)$: ϕ doit être vrai dans au plus un état du plan;
- $(sometime\ -\ after\ \phi\ \psi)$: à chaque fois que ϕ est vrai dans un état *s*, il doit y avoir un état *s'* égal ou après *s* où ψ est vrai;
- $(sometime\ -\ before\ \phi\ \psi)$: à chaque fois que ϕ est vrai dans un état *s*, il doit y avoir un état *s'* avant *s* où ψ est vrai;
- $(always\ -\ within\ t\ \phi\ \psi)$: à chaque fois que ϕ est vrai dans un état s, il doit y avoir un état *s'* après *s* où ψ est vrai et qui est au plus, à *t* pas après *s*.

Une norme LTL est violée si son interprétation est fausse dans un plan donné. Ces deux types de normes diffèrent dans leur vérification: les normes conditionnelles peuvent être vérifiées dans un seul état, tandis que les normes LTL doivent être vérifiées sur une séquence d'états finie.

1.7.2.2. Technique de planification utilisée

Le générateur de plans d'actions se base sur l'algorithme de Graphplan [Blum & Furst, 1997], qui est un POP (Planificateur à ordre partiel) avec un graphe comme structure de données représentant le plan. Ce graphe de planification est un graphe directionnel qui présente plusieurs niveaux, alternant entre un niveau d'actions et de propositions, sur lequel est effectuée une recherche en chaînage arrière. Graphplan maintient également la liste des propositions et des actions qui sont mutuellement exclusives.

Les arcs reliant les niveaux adjacents sont de trois types :

a) un arc de précondition : entre un niveau d'action et un niveau de proposition qui le précède; la représentation d'actions reste identique à celle de STRIPS.
b) Un arc de conséquence : entre un niveau d'action et un niveau de proposition qui lui succède;
c) Un arc de maintenance : un arc vide qui signifie que la proposition est maintenue pour le niveau de proposition suivant.

Graphplan maintient également à chaque niveau la cohérence en détectant les actions et les propositions qui sont mutuellement exclusives (abrégé en *mutex*).

Définition *(Actions mutex) : deux actions de même niveau sont mutex s'il y a :*

- *Effets incohérents : une des actions annule l'effet de l'autre ;*
- *Interférence : une action détruit la précondition de l'autre ;*
- *Besoins concurrents : les deux actions ont des préconditions mutex.*

Définition *(Propositions mutex) : deux propositions de même niveau sont mutex s'il y a :*

- *L'un est la négation de l'autre ;*
- *Toutes les actions qui permettent de les établir sont mutex par pair*

Le Code 4 représente le processus d'extraction de solution de Graphplan dans ses grandes lignes.

```
Plan solutionExtraction(Goal goal, int level, boolean isViolationPlan, String type){
  if (level == 0){
    if (type.equals("Norm Violating") && !isViolationPlan){
      backtrack();
    }
    return solution;
  }

  List<Action> chosenActions = new ArrayList<>;

  for(Atom atom : goal.getAtoms()){
    chosenActions.add(getActionToAchieve(atom));
  }

  if (chosenActions.containsMutexActions()){
    backtrack();
  }

  Goal newGoal = allPrecondtions(chosenActions);
  boolean newIsViolationPlan = isViolation(chosenActions, newGoal);
  if (type.equals("Norm Compliant") && newIsViolation) {
    backtrack();
  }

  return solutionExtraction(newGoal, level-1, newIsViolationPlan, type);
}
```

Code 4: Algorithme du chaînage arrière chargé d'extraire le plan solution dans Graphlan

Selon le vocabulaire de planification que nous avons posé pour comparer les techniques de planification, Graphplan peut être décrit comme suit :

- *State* : un graphe de planification étendu jusqu'à un certain niveau;
- *Operator* : expansion du graphe de planification;
- *getOptions* : pour un niveau *i*, noté *niv(i)* les expansions possibles du graphe sont l'ajout de nouveaux niveaux d'actions α applicables dans *niv(i)* tel que $préconditions(\alpha) \subseteq niv(i)$. L'ajout de ce niveau entraîne automatiquement la création d'un niveau de propositions, obtenues à travers les préconditions de l'action.
- *isFinal* : un graphe dont la phase de chaînage arrière établit une relation entre les différents niveaux, du premier au niveau qui contient les objectifs, sans qu'il y ait de relation mutex;
- *isValid* : il existe des propositions ou des actions mutex dans le même niveau;
- *evaluateState* : le nombre de préconditions non-résolues;
- *evaluateOperator*: non spécifié, mais peut être affecté arbitrairement à 1 .

1.7.2.3. Stratégies de prise en compte des normes

Pour prendre en compte les normes dans Graphplan, les auteurs proposent un algorithme qui vise à exclure toute solution qui viole les normes : pour cela, l'extraction de la solution récupère l'ensemble des solutions au niveau actuel, et itère sur chaque solution pour vérifier si elle viole ou respecte la norme. Graphplan effectue deux recherches dans un espace d'états pour prendre en compte les normes :

1) Une recherche en chaînage arrière pour les normes conditionnelles;
2) Une recherche en chaînage avant pour les normes LTL.

Cette recherche en chaînage avant pour vérifier les normes LTL se fait grâce à un algorithme UCS (*Uniform-Cost Search*) qui est modifié pour prendre en compte les normes : elle permet de minimiser le coût total des actions et des pénalités des violations de normes. Pour garantir un plan qui prend en compte les normes, on vérifie si le plan à ordre partiel viole une norme, puis on teste si l'état est un état solution (*isFinal*), et enfin on ajoute l'état successeur dans la frontière.

Au terme de cette vérification, deux concepts sont ajoutés dans l'optique de pouvoir qualifier les plans à ordre partiel par rapport aux normes. Un plan à ordre partiel peut donc présenter :

1) Une violation absolue : le plan ne présente aucune possibilité pour se conformer aux normes à nouveau;
2) Une violation courante : le plan viole actuellement quelques normes mais il reste possible qu'elle se conforme aux normes plus tard.

Si le but est d'obtenir seulement des plans conformes aux plans :

- Les plans présentant une violation absolue;
- Les plans présentant une violation courante sont ajoutés à la frontière, mais ne peuvent pas être testés comme plan solution (*isFinal*).

Si le but est d'obtenir des plans qui peuvent violer les normes :

- Tous les états sont ajoutés à la frontière;
- Seuls les plans avec une violation courante peuvent être testés comme plan solution.

Dans les deux cas, l'état solution doit minimiser le coût total des actions et des violations de normes. Pour identifier les violations absolues et les violations courantes :

1) En créant un plan à ordre total avec les actions dans chaque niveau, sur lequel nous pouvons vérifier d'éventuelles violations de normes;
2) En associant à chaque état les informations sur les normes violées, et les revérifiant après chaque opérateur qui ajoute une action.

Pour synthétiser les travaux de Meneguzzi et al sur leur planification :

- Seules les interdictions et les obligations sont prises en compte, les permissions ne sont pas prises en compte;
- Les auteurs distinguent deux types de normes, les normes conditionnelles qui représentent des normes applicables sur un seul état à la fois, et les normes LTL qui sont des obligations s'appliquant sur une intervalle de situation avec les modalités temporelles telles que : *at end, always, at most once, et always within*.
- Les auteurs utilisent plusieurs planificateurs pour traiter différents types de normes (normes conditionnelles ou normes LTL), leur apport principal réside dans le fait qu'ils permettent également de produire des plans qui violent les normes tout en minimisant le coût du plan, qui doit être quantifié.

1.7.3. DIARC & ADE

L'architecture hybride délibérative-réactive DIARC (Distributed, Integrated, Affect, Reflection, Cognitive) est une architecture conçue pour des robots capables de raisonner sur des valeurs morales, y compris les normes sociales [Jackson et al., 2021] implémentées dans ADE (Agent Development Environment). Le générateur de plans d'actions utilisé par DIARC se base sur une représentation symbolique en PDDL3 qui permet notamment de spécifier des

prédicats qui doivent toujours rester vrais durant la planification, et que [Jackson et al., 2021] utilisent pour encoder les normes.

1.7.3.1. Représentation des normes

Formellement, une norme, dite "une norme morale" dans DIARC est représentée par la formule $C \Rightarrow op(x)$ où : C est un contexte, op est une modalité déontique (*permise, interdite, obligatoire*), et x est un ensemble d'actions ou de propositions. Cette formule exprime donc que telle action ou atome est interdite, permise, ou obligatoire dans un certain contexte.

Les normes sont ensuite converties en tant que contraintes vers le domaine de planification de PDDL. Par exemple, dans le cas des obligations et des interdictions, elles peuvent se transformer en des contraintes qui doivent être respectées à chaque étape du plan. Cette transformation est décrite à travers le Tableau 4.

Tableau 4: Conversion des normes en contraintes de planification en PDDL

Norme morale	Contrainte de planification	Signification
$C \Rightarrow obligated(x)$	$\forall k, \neg (C^{(k)} \wedge \neg x^{(k)})$	Si x est obligatoire, alors pour tout état k du plan, $\neg x$ ne doit pas être satisfait.
$C \Rightarrow forbidden(x)$	$\forall k, \neg (C^{(k)} \wedge x^{(k)})$	Si x est interdit, alors pour tout état k du plan, x ne doit pas être satisfait.

1.7.3.2. Technique de planification utilisée

DIARC utilise un générateur de plans d'actions basé sur des contraintes [Kautz & Selman, 1999]. Les descriptions des normes sont traduites en un ensemble de formules booléennes qui doivent être satisfaites à chaque étape du plan. Pour trouver comment le plan

peut satisfaire ces contraintes, DIARC utilise un solveur SMT (Satisfiability Modulo Theories) [Barrett et al., 2015; De Moura & Bjorner, 2011].

1.7.3.3. Stratégies de prise en compte des normes

Pour incorporer les normes morales, les contraintes booléennes qui résultent de la traduction des normes font que le planificateur retourne uniquement des plans qui satisfassent les normes morales.

Lorsque le solveur ne trouve pas de plan, le générateur de plans d'actions produit un noyau de non-satisfiabilité[11] qui peut être analysé pour comprendre la raison de l'échec de la planification: il s'agit de l'ensemble des clauses (objectifs, normes, et préconditions d'actions) qui rendent la planification insatisfaisable. Si ce noyau de non-satisfiabilité contient à la fois une action et une norme, on peut en déduire que la norme et les actions sont mutuellement exclusives, et que les actions sont donc à l'origine de l'échec de la planification avec les normes. Pour fournir à DIARC les actions nécessaires dans l'accomplissement d'un objectif, le gestionnaire d'objectifs (Goal Manager) communique avec le planificateur en fournissant :

1) Les actions possibles avec leurs arguments, préconditions, et postconditions;
2) Les prédicats qui servent à décrire 1) les préconditions, postconditions des actions quand elles sont fournies au planificateur, 2) le contexte dans lequel l'agent avec le prédicat `in(?context)`, et 3) les propriétés et le type de certains objets, e.g. *"box(x)", "forest(x)"* , mais également des informations sur les autres agents;
3) Les objets du système sur lesquels les prédicats peuvent s'appliquer, et les actions peuvent s'exécuter;
4) Les conditions initiales de la planification à chaque fois qu'un plan est demandé, qui est l'équivalent d'une situation initiale;
5) Les objectifs à accomplir.

En retour, le Goal Manager peut recevoir du planificateur soit un plan solution, soit un noyau de non-satisfaisabilité. Ils prennent en compte les trois modalités déontiques :

[11] Traduit littéralement de l'anglais original "unsatisfiable core"

obligation, permission, et interdiction. Pour la prise en compte des normes, son principal apport est que :

- L'agent va devoir se conformer à toutes les normes, en s'assurant de lui donner toutes les informations nécessaires;
- Si le planificateur de l'agent échoue, un noyau de non satisfaisabilité est généré pour dire quelle norme empêche l'agent de trouver un plan, ou encore quel objectif et quelle norme sont mutuellement exclusives;

1.7.4. Doms de Maia et Sichman

Se focalisant sur l'autonomie de l'agent pour former des plans de zéro, [Maia & Sichman, 2016] propose un planificateur normatif basé sur HTN en raison de ses similarités avec l'implémentation typique de l'architecture BDI. Le tout est implémenté sur le framework JaCaMo de [Boissier et al., 2011].

1.7.4.1. Représentation des normes

Les normes dans DomS se basent sur la notion d'organisation, puisque les auteurs utilisent le modèle MOISE pour représenter :

- La spécification structurelle incluant les rôles, les groupes et les liens;
- La spécification fonctionnelle qui permet d'accomplir les objectifs organisationnels qui permet de décomposer les objectifs en sous objectifs. La notion de mission est utilisée pour spécifier un ensemble d'objectifs et de sous objectifs que l'agent doit accomplir;
- La spécification déontique pour spécifier les relations entre les rôles et les missions, par exemple il est possible de dire qu'une mission m1 est obligatoire pour tel rôle.

Puisque MOISE ne dispose pas d'éléments pour fournir à l'agent une autonomie en termes de planification[12] en plus de ces spécifications organisationnelles, les auteurs proposent d'ajouter un planificateur HTN.

1.7.4.2. Planificateur utilisé

Pour créer des plans, les auteurs se basent sur HTN qui vont décomposer une tâche complexe, en tâches primitives à l'aide de méthodes. Les auteurs ajoutent une autre spécification appelée Domain Specification (DomS), contenant la traduction des objectifs organisationnels de MOISE en deux types d'objectifs : des objectifs déclaratifs (pour accomplir des états du monde) , et des objectifs procéduraux (pour accomplir une certaine action).

1.7.4.3. Stratégie de prise en compte des normes

Les normes spécifiées par les objectifs organisationnels de MOISE sont accomplies en les traduisant sous la forme d'un problème de planification HTN, et en les divisant en objectifs déclaratifs ou objectifs procéduraux. Initialement, MOISE spécifie un lien d'autorité entre les agents, qui permet à un agent d'imposer un comportement à un autre agent. Les auteurs changent ce lien d'autorité, en le remplaçant par un planificateur automatisée.

Cette approche offre donc un moyen autonome pour l'agent de satisfaire un objectif normatif , mais sans aborder la notion de conflit entre les normes et les objectifs de l'agent. Le planificateur reçoit tout simplement les objectifs normatifs depuis le modèle MOISE et les exécute.

[12] L'autonomie en termes de planification est introduite par [Castelfranchi, 2000] comme étant la capacité de l'agent à créer, exécuter, modifier ou choisir un plan pour accomplir un objectif.

1.7.5. Kammler et al.

[Kammler et al., 2022] utilisent la notion de valeurs pour prendre en compte les normes. Les valeurs représentent des mécanismes d'évaluation des actions et des événements qui rendent un certain état désirable ou non: l'agent cherche alors à promouvoir les états qui sont en alignement avec ses valeurs.

1.7.5.1. Représentation des normes

Les auteurs utilisent une extension de la grammaire institutionnelle ADICO nommée ADICDlRO[Kammler et al., 2021] qui introduit la notion de délai pour accomplir l'objectif d'une norme (D1) et les actions pour réparer la violation d'une norme (R). Les normes auront donc ainsi les composants ADICO classiques et traiteront des obligations, des interdictions et des permissions, mais proposeront pour l'objectif de la norme (I) : l'objet sur lequel l'objectif doit se porter, et l'action qui doit être fait sur cet objet.

L'addition la plus importante ADICDlRO est que les normes peuvent promouvoir ou dégrader certaines valeurs en fonction de leur objectif. Par exemple, restreindre le nombre de tables durant une pandémie peut promouvoir la valeur de la sécurité, mais dégrader la valeur de d'accomplissement et de pouvoir.

Pour leur système de valeurs, les auteurs se basent sur les 10 valeurs dans [Schwartz, 2012] que sont : l'autodirection, la stimulation, l'hédonisme, l'accomplissement, le pouvoir, la sécurité, la conformité, la tradition, la bienveillance et l'universalisme qui seront classés par priorité selon l'agent [Heidari et al., 2020]. Les valeurs servent de moyen d'évaluation pour les actions et les événements, qui vont motiver l'agent à préférer certains états par rapport à d'autres. Un agent aura donc une perspective associant un ordre total de priorité (dit PrioV, ou *priority of values*) sur les dix valeurs tel que $V = \{V1, ..., V10\}$ où chaque valeur aura un indice d'importance différent variant entre 0 et 1.

Leurs approches combinent différents niveaux de délibération leur permettant de choisir un plan parmi une base de données de plans (simple), de gérer des situations où des

actions ne sont plus exécutables et qu'il faut trouver des actions alternatives (moyen), et, si nécessaire, de faire appel à un planificateur ou d'aborder de nouveaux objectifs (complexe).

1.7.5.2. Planificateur utilisé

Les auteurs utilisent une variante de STRIPS appelée GOAP (Goal-Oriented Action Planning) qui se base sur un chaînage arrière [Orkin, 2006]. La planification se fait par des modèles de plans [Dignum, 2018] au lieu d'actions atomiques.

Formellement, un modèle de plan décrit une séquence d'actions à l'aide de points de contrôle. Un point de contrôle est un point fixe dans un plan qui doit être accompli à un certain moment du plan [Kammler et al. 2022a], par exemple, à un moment donné du plan, il faudra payer les taxes au sein d'un village quand on est membre du village.

1.7.5.3. Stratégies de prise en compte des normes

En tant que planificateur de type chaînage arrière, GOAP se commence par l'état final et avance vers l'état initial:

1. Le planificateur vérifie si le dernier point de contrôle de chaque séquence d'actions remplit l'objectif;
2. Le planificateur va ensuite chercher à atteindre le point de contrôle qui le précède;
3. Ce processus sera répété jusqu'à en revenir à la situation initiale de l'agent.

Pour ajouter la notion de valeur aux objectifs de l'agent, les auteurs utilisent la notion de jauge[13] pour représenter les valeurs que l'agent souhaite satisfaire, dont une jauge pour la notion de conformité (aux normes et aux comportements des autres agents).

Dans cette approche, les normes sont rattachées à des valeurs qui peuvent promouvoir, ou aller à l'encontre des valeurs que l'agent désire. De ce fait, l'agent peut utiliser la priorité de la valeur rattachée aux normes pour la violer ou la prendre en compte.

[13] En anglais : watertank

Dans l'essentiel, pour prendre une décision sur la prochaine action souhaitée, l'agent selon [Kammler et al., 2022] regarde d'abord si elle est interdite par une norme. Si la norme promeut les mêmes valeurs que celles qui sont importantes pour l'agent, il s'y conformera, et pourra générer un comportement par planification; sinon, il compare le coût de la prise en compte de la norme à la norme au coût de la violation de la norme. La sélection est alors basée sur une fonction argmax, les *PrioV* qui jouent le rôle de multiplicateurs. Elle nécessite non seulement de décrire quelles valeurs sont plus importantes pour l'agent mais aussi d'associer à chaque norme une valeur et leur contribution quantitative à cette valeur.

1.8. Synthèse de l'état de l'art

Pour avoir une vue synthétique de la revue de la littérature, nous pouvons résumer les différentes architectures normatives et leurs apports sur la prise en compte des normes dans le Tableau 5 avec les dimensions suivantes :

- Modalité déontique : quelles sont les modalités déontiques prises en compte ?
- Objectifs des normes : quels types de concepts sont rendus obligatoires, interdites, ou permises, des actions et/ou des états du monde ?
- Stratégies de prise en compte des normes : par quels moyens l'agent prend en compte les normes,
    - est ce par une une fonction d'utilité pour privilégier certains objectifs ou normes (=utilitaire) ?
    - par un typage des agents ou de leurs comportements (=Typage) ?
    - par un critère pour écarter des comportements incohérents à une norme (= Filtration) ?
    - par l'élaboration d'un nouveau plan (= Planification) ? ou autre ?
- Planification automatisée : est ce que l'agent est autonome en termes de planification?
- Sanction et récompenses : est ce que les sanctions et les récompenses sont représentées et/ou utilisées quand l'agent engendre un comportement ?
- Raisonnement quantitatif : est ce qu'il est nécessaire de quantifier les actions, et les situations ? Ce type de raisonnement ne peut fonctionner que si l'on est capable de quantifier les interactions de l'agent avec son environnement.

Tableau 5: Résumé synthétique de l'état de l'art sur les architectures normatives

Travail	Modalité déontique	Objectifs des normes	Stratégies de prise en compte des normes	Planification automatisée	Sanction et récompenses	Raisonnement quantitatif
Panagiotidi et al., 2013	Obligation, Interdiction	État	Réparation : chaque violation introduit la nécessité de la réparer	OUI	NON	OUI
Kammler et al., 2022	Obligation, Interdiction, Permission	Action État	Planification avec des valeurs	OUI (GOAP)	OUI	OUI (Sur les jauges)
Maia & Sichman, 2020 (Doms)	Obligation Permission	Action État	Régimentation : les objectifs de la norme doivent être accomplis	OUI (conversion vers HTN)	NON	NON
Broersen et al., 2001 (BOID)	Obligation	État	Typage fixe : type des agents	NON	NON (optionnelle)	NON (optionnelle)
Lopez & Marquez, 2004	Obligation, Interdiction	État	Typage fixe : type des agents	NON	OUI	OUI
Viana et al., 2015 (JSAN)	Obligation, Interdiction, Permission	Action État	Typage fixe : type d'agent extensible par des classes Java	NON (Plans prédéfinis)	OUI	OUI (sur les récompenses et les sanctions)
Andrighetto et al., 2010 (EMIL-A)	Obligation, Interdiction, Permission	NON-SPECIFIÉ	Utilitaire : par la saillance des normes	NON-SPECIFIÉ	NON-SPÉCIFIÉ	OUI

Travail	**Modalité déontique**	**Objectifs des normes**	**Stratégies de prise en compte des normes**	**Planification automatisée**	**Sanction et récompenses**	**Raisonnement quantitatif**
Kollingbaum & Norman, 2003 (NoA)	Obligation, Interdiction, Permission	Action État	Filtration: Maximisation de la cohérence de l'ensemble des normes	NON (Plans prédéfinis)	NON	NON
Meneguzzi & Luck, 2009 (Normative AgentSpeak(L))	Obligation, Interdiction	Action État	Modification de la bibliothèque de plans Planification et libraire de plans	OUI (STRIPS)	NON	NON
De Campos et al., 2012	Obligation, Interdiction	Action, État	Filtration : choisir l'action obligatoire non-interdite, si elle est interdite, choisir l'action ni interdite, ni obligatoire.	NON	OUI	NON (excepté le temps)
dos Santos Neto et al., 2012 (ANA)	Obligation, Interdiction	État	Filtration : par critère de motivation de l'agent par rapport aux sanctions, et aux récompenses.	NON-SPÉCIFIÉ (S'arrête aux intentions)	OUI	OUI
Criado et al., 2010 (n-BDI)	Obligation, Interdiction	État	Typage et Utilitaire: typage de l'agent et évaluation de la (in)désirabilité des normes	NON	OUI	OUI
Jackson et al., 2021 (DIARC ADE)	Obligation, Interdiction, Permission	État Action	Régimentation : trouver un plan qui satisfait toutes les normes, et informer pourquoi un plan n'a pas pu être trouvé en cas d'échec.	OUI (SMT)	NON	NON

De ce tableau et des différents chapitres de l'état de l'art, nous pouvons tirer les conclusions suivantes :

(1) Les institutions à travers ses normes régulatrices et constitutives, et son ontologie décrivent des normes qui seront affectées à des rôles qui seront joués par les individus et les objets du système; En affectant des objets et des agents concrets aux concepts des institutions, nous obtenons la notion d'organisation. Une des premières grammaires institutionnelles pour décrire les normes est présentée par [Crawford & Ostrom, 1995] avec le formalisme ADICO, qui présente plusieurs extensions pour enrichir son expressivité; toutefois, les thématiciens se posent des questions sur les impacts de ces institutions sur le comportement des individus afin d'évaluer si elles sont efficaces ou non;

(2) Pour pouvoir évaluer les impacts des normes de manière explicable, des modèles à base d'agents sont explorés qui peuvent engendrer des comportements autonomes, grâce à une architecture d'agent, y compris avec les normes. Dans l'état de l'art sur les architectures d'agent, on utilise différentes façons d'engendrer des comportements autonomes. Nous en tirons qu'en fonction de la complexité des comportements souhaités, nous pouvons adopter des architectures réactives, plus simples, ou bien des architectures plus sophistiquées telles que les architectures délibératives ou cognitives. Au vu de la prise en compte des normes, les architectures délibératives comme l'architecture BDI semblent présenter un meilleur équilibre entre explicabilité et expressivité que les autres architectures;

(3) Il existe en effet une très vaste littérature sur les architectures d'agent normatives, où l'on ne retrouve pas toujours la notion d'institution et d'organisation. Généralement, plusieurs de ces architectures se basent sur une stratégie fixe pour déterminer si l'agent se conforme à la norme ou la viole, telle que le typage des agents ou une stratégie prédéfinie par le modélisateur (égoïste, rebelle, social…), pour définir les préférences sociales des agents.

(4) D'autres stratégies se basent sur des fonctions d'utilité ou une quantification de la motivation des agents pour plus de flexibilité, mais quantifier les apports de la prise en compte de chaque norme, de chaque violation, de chaque récompense, et de chaque sanction n'est pas toujours faisable.

(5) Du côté des implémentations opérationnelles, la grande majorité des langages et outils orientés-agents incluant les normes se voient simplifiées par l'utilisation d'une bibliothèque de plans pré-compilés pour plus de praticité, alors que les agents se veulent être des systèmes ouverts, avec potentiellement de nouvelles normes.

Nous pouvons donc déduire de cet état de l'art que les normes ont besoin de mieux être prises en compte dans les comportements des agents de sorte à pouvoir.

1) produire des comportements autonomes en réponse aux normes, et non conditionnés entièrement par un ensemble de plans prédéfinis par le modélisateur;
2) convenir aux environnements ouverts des SMA dans un contexte multi-institutionnel où plusieurs agents de différents types et implémentés différemment peuvent interagir entre eux et avec plusieurs normes provenant de différentes organisations qu'ils doivent pouvoir prendre en compte;
3) et fournir un comportement explicable de sorte à comprendre les raisons derrière le comportement produit, en décrivant comment les obligations, les interdictions, et les permissions ont impacté sur le comportement des agents.

2. Contributions

Au vu de l'état de l'art et des objectifs posés, nous pouvons maintenant situer notre contribution et la décrire plus en détail. Pour remédier aux problèmes constatés dans l'état de l'art et améliorer la prise en compte des normes, nous proposons de concevoir un outil de modélisation prenant en compte les normes :

- Avec la notion d'institution et d'organisation conformément à la littérature;
- Sans la nécessité de se baser constamment sur un raisonnement quantitatif qui n'est pas toujours faisable;
- Avec un planificateur engendrant le comportement prenant en compte les normes;
- Avec une architecture d'agent dotée d'un langage dédié pour pouvoir définir les concepts nécessaires pour pouvoir spécifier son comportement.

Pour prendre en compte les normes, il nous faut non seulement une architecture d'agent, mais aussi une spécification de la manière dont l'agent peut générer de nouveaux plans, contraints par les normes applicables, et surtout expliquer pourquoi tel comportement a été choisi par l'agent. Ce chapitre "Contributions" se divisera en 4 sections :

1) Une présentation des outils utilisés;
2) Une définition des concepts utilisés pour décrire les normes, les institutions, les organisations, et pour décrire les objets, les actions, et les situations dans le langage proposé;
3) Une présentation des extensions proposées pour pouvoir intégrer ces concepts dans un problème de planification;
4) Une présentation des extensions proposées détaillant comment ces concepts intégrés sont utilisés pour engendrer un comportement normatif.

2.1. Outils utilisés

2.1.1. Architecture d'agent délibérative

Parmi les architectures d'agent présentées dans l'état de l'art, nous devons choisir quel type d'architecture utiliser pour prendre en compte les normes. Les architectures réactives, de par leur nature, ne sont pas adaptées pour générer ce type de comportements complexes qui savent comment respecter ou violer une norme dans une certaine situation. D'un autre côté, les architectures cognitives telles que SOAR, ACT-R ou CLARION peuvent y parvenir, mais en raison de leur complexité apparente, il serait difficile d'en faire un outil accessible pour les thématiciens, et les modélisateurs.

À travers la littérature, nous pouvons remarquer que l'architecture BDI est l'architecture délibérative la plus communément adoptée dans la communauté SMA, et qu'elle est à l'origine de plusieurs extensions normatives. [Adam et al., 2011] mentionne que l'architecture BDI constitue l'une des architectures qui reflètent un vocabulaire à la fois accessible pour des non-informaticiens, tout en étant pertinent pour de la simulation sociale, incluant des comportements complexes.

Nous choisissons de créer nos contributions, de sorte qu'elles puissent être intégrées dans des architectures délibératives telles que l'architecture BDI, potentiellement combinées avec des couches réactives pour produire une architecture d'agent hybride. Pour obtenir des plans qui respectent les normes, nous devons choisir un planificateur pour générer les plans.

2.1.2. Planificateur à ordre partiel (POP)

Pour engendrer un comportement qui prend en compte les normes, un planificateur est nécessaire. Nous devons choisir un paradigme, et l'étendre avec les notions de normes. Nous pouvons entre autres choisir :

- Une planification classique en chaînage avant, ou en chaînage arrière ou avec un planificateur à ordre partiel;
- Une planification hiérarchisée de tâches (HTN);

- Une planification sous la forme d'une chaîne de Markov;
- Une planification par algorithme génétique;

Pour pouvoir engendrer le comportement de l'agent, nous utilisons un planificateur à ordre partiel classique (POP), au vu des arguments présentés dans 1.6.7. sur sa flexibilité et sa stratégie de moindre engagement qui sont propices pour la prise en compte des normes.

Pour le cas de la prise en compte des normes, il nous faut notamment savoir : quelles normes sont applicables ? quelles normes ont été applicables dans les situations précédentes, mais ne le sont plus dans la situation actuelle ? Il nous faut donc raisonner non pas sur une situation, mais sur une séquence de situations.

Dans le cas où l'on se décide à prendre les planificateurs en chaînage avant, il faudrait toutefois implémenter un algorithme récursif pour vérifier les normes applicables aux situations passées à l'instar de [Panagiotidi et al., 2013], ou bien recalculer les normes applicables à chaque situation, ce qui introduit une complexité au planificateur, pour revenir à ce que fait exactement le POP.

Le chaînage arrière, quant à lui, peut s'avérer difficile à implémenter puisque les états finaux sont décrits par des contraintes plutôt que par des atomes explicites. Notamment, il est difficile de s'accorder sur comment générer l'ensemble des prédécesseurs possibles de l'ensemble des états finaux.

Face à ces critères, en plus des arguments déjà avancés dans la section 1.6., et des arguments présentés dans [Kvarnstrom, 2011] : le POP est plus adapté à de la planification pour des agents. En effet, le POP raisonne sur l'espace des plans possibles et peut retracer dans quelles situations une norme est active, et quand est-ce qu'elle ne l'est pas ou ne l'est plus. Son ordre partiel et ses contraintes de (non) codénotation en font un planificateur à moindre engagement, qui permettrait à l'agent d'adapter son comportement aux normes sans se fixer forcément ni sur un plan fixe (totalement ordonné), ni sur un objet d'action en particulier, puisque les variables du plan ne codénotent pas forcément sur des constantes.

2.2. Ingénierie dirigée par les modèles

Pour pouvoir permettre au modélisateur d'exprimer un problème de planification dans un contexte multi-institutionnel et multi-organisationnel avec les normes, nous utilisons l'Ingénierie dirigée par les modèles (IDM) avec trois (03) artéfacts :

1) une syntaxe abstraite qui décrit les concepts de base du langage et leurs relations dans un méta-modèle;
2) une syntaxe concrète qui décrit comment créer des instructions correctement avec les concepts décrits par la syntaxe abstraite;
3) une sémantique opérationnelle pour décrire le sens de ce qui a été écrit.

Dans ce chapitre, nous détaillons les syntaxes abstraites et concrètes de ces concepts, avec 3 couches de conversion (Figure 25) avec : 1) Un problème de planification, avec la représentation des actions, des objectifs, des normes et de la situation initiale en un ensemble d'états et d'opérateurs; 2) Une recherche dans un espace d'états pour produire le plan solution; et 3) Une retranscription d'un plan solution vers l'utilisateur.

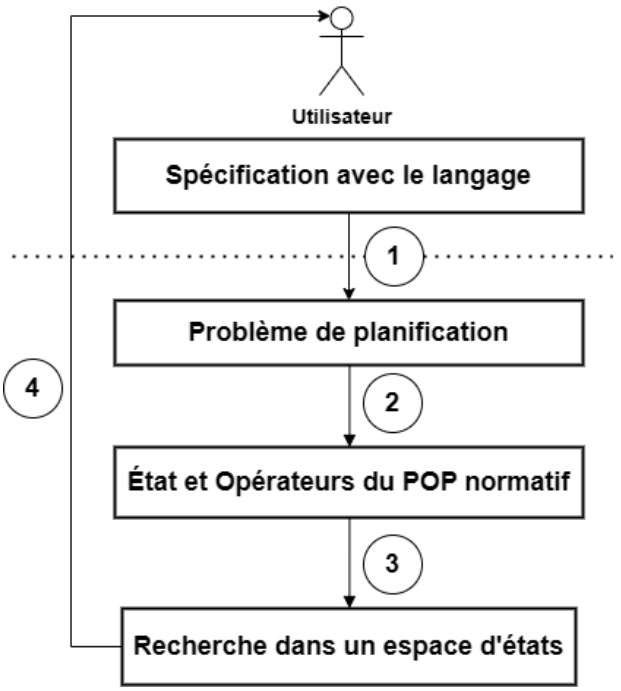


Figure 25: Les différentes étapes de traduction de la spécification du problème de planification à l'obtention du plan

2.2.1. Syntaxe des prédicats et des atomes

Pour décrire formellement ces connaissances, nous nous basons sur un fragment de la logique de prédicats du premier ordre avec des termes reliés par un ET logique dans la section 1.5.1.2 mais sans inclure les termes fonctionnels. Cette syntaxe abstraite des atomes est représentée par la Figure 26

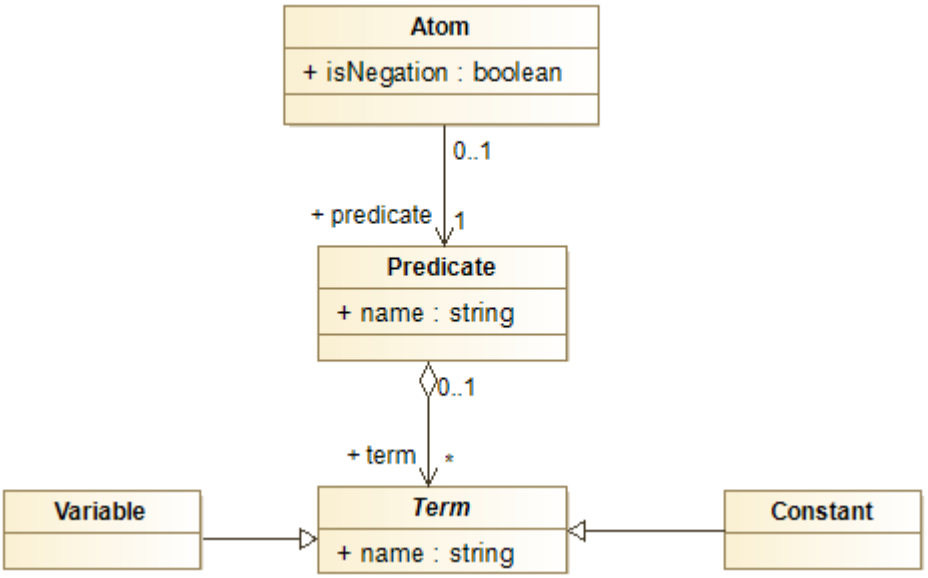


Figure 26 : Syntaxe abstraite des atomes et des prédicats

La syntaxe concrète d'un atome peut se traduire par le Code 5 avec un prédicat, préfixé potentiellement par un mot-clé "*not*" dans le cas où il s'agit d'une négation. Cette syntaxe concrète est décrite avec la méta-syntaxe EBNF (Extended Backus-Naur Form) [Wirth, 1977], typiquement utilisée pour spécifier la grammaire d'un langage en raison de son format compact et de sa simplicité d'expression.

```
Atom ::= ['not'] <Predicate>
```

Code 5 : Syntaxe concrète d'un atome

Quant à la syntaxe concrète d'un prédicat, elle est représentée par le nom du prédicat, suivi de ses termes entre parenthèses sur le Code 6 (1). Les constantes désignent des objets concrets du système qui seront typés par des rôles.

Pour pouvoir distinguer les constantes des variables, nous ajoutons en tant que préfixe le mot clé "*const*", et les variables avec le mot clé "*var*" comme démontré sur le Code 6 (2) et (3).

```
(1)   Predicate ::= <Predicate.name> '(' {<Term>} ')'
(2)   Variable ::= 'var' <Variable.name>
(3)   Constant ::= 'const' <Constant.name>
```

Code 6 : Syntaxe concrète d'un prédicat et de la déclaration de ses termes

2.2.2. Syntaxe abstraite du problème de planification normatif

Tout d'abord, pour pouvoir prendre en compte les normes spécifiées par le langage, nous devons les intégrer à la définition même du problème de planification classique. Un problème de planification classique est décrit par : une situation initiale, un objectif, et un répertoire d'actions, tel que le démontre la Figure 27.

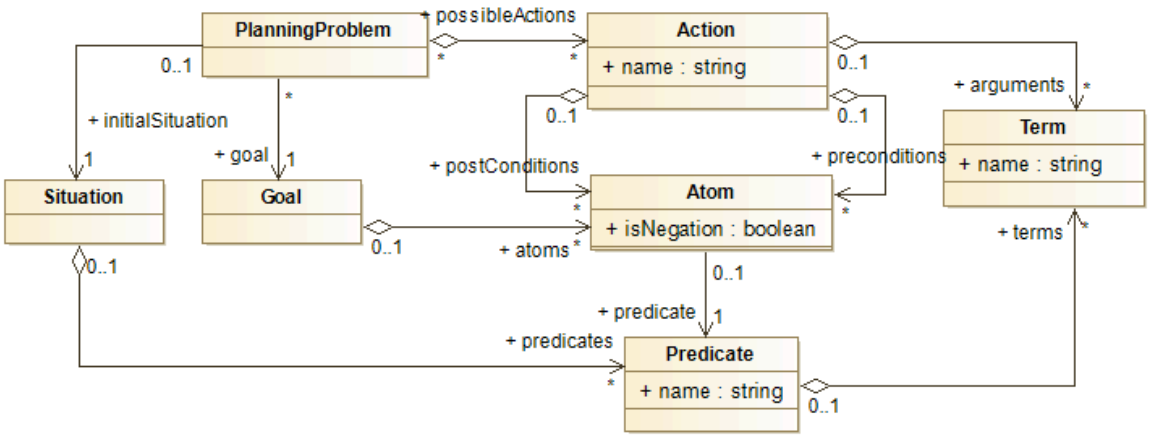


Figure 27: Syntaxe abstraite d'un problème de planification classique

Pour prendre en compte les normes, nous ajoutons un ensemble d'organisations, qui sont des instances d'institutions. Une vue synoptique des extensions apportées au problème de planification, avec la notion d'institutions et d'organisations est illustrée sur la syntaxe abstraite de la Figure 28 [Ramarozaka et al., 2022].

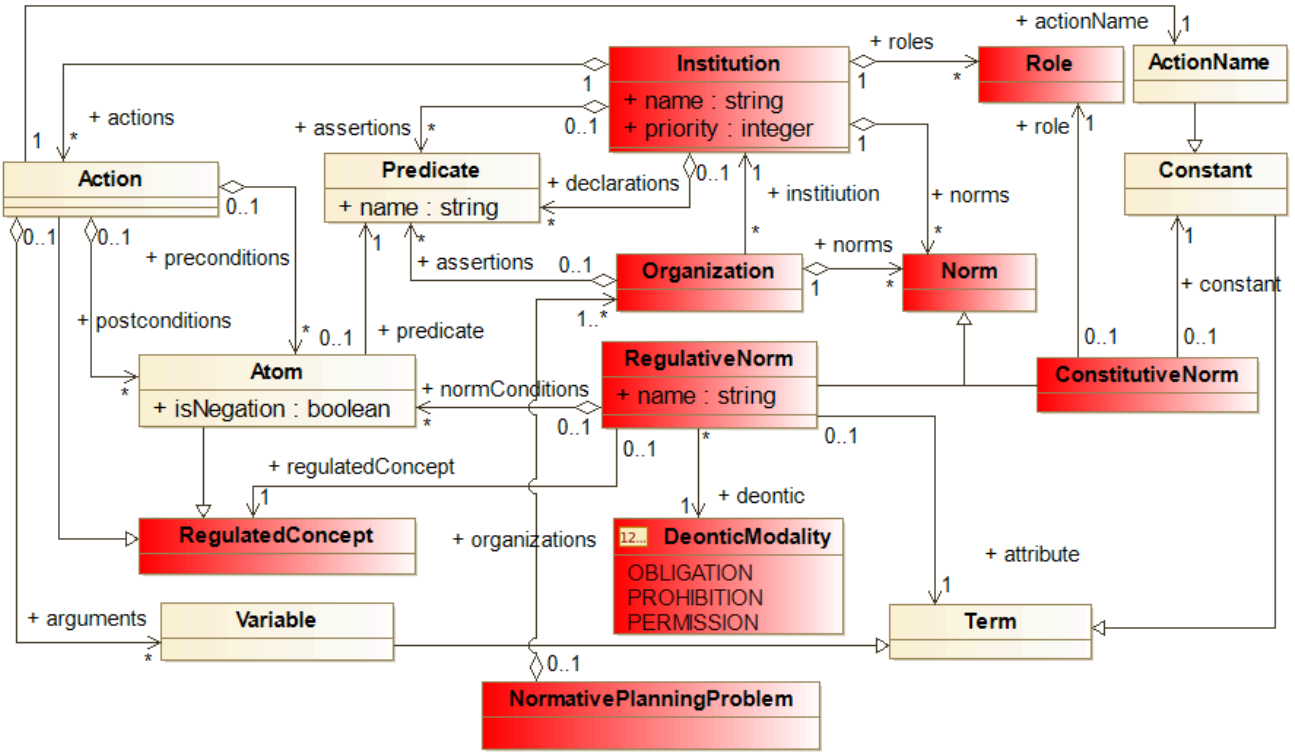


Figure 28: Syntaxe abstraite d'un problème de planification normatif avec nos contributions (en rouge)

Pour décortiquer ces contributions, nous commençons par décrire : 1) les concepts qui traitent des normes (les institutions, les organisations, les rôles, les normes régulatrices, les normes constitutives), puis 2) comment elles modifient la définition du problème de planification classique.

2.2.3. Syntaxe des institutions

Une institution se compose d'un ensemble de rôles, d'assertions, de normes régulatrices, de normes constitutives et d'actions. Cette section décrit successivement ces concepts ainsi que leurs syntaxes.

2.2.3.1. Syntaxe des rôles

Les rôles vont permettre de décrire le type d'un ensemble d'objets nommés par des constantes. Par exemple, un rôle peut décrire que la constante RAKOTO est de type agent, ou qu'une constante FIANARANTSOA est de type ville. Comme le décrit la Figure 28, une institution décrit plusieurs rôles. Pour décrire un rôle avec le langage proposé, le Code 7 décrit sa syntaxe avec EBNF avec tout le nom du rôle avec le préfixe "*Role*" par exemple : *Role hunter*.

```
Role ::= 'Role' <Role.name>
```

Code 7: Syntaxe concrète d'un rôle

2.2.3.2. Syntaxe des assertions et des déclarations

Les assertions servent à décrire des situations concrètes relatives à l'institution. En termes de syntaxe abstraite, les assertions sont donc représentées par un ensemble de prédicats rattachés à une institution sur la Figure 28. Dans le cadre des logique descriptives, ces assertions qui se portent sur des individus correspondent à la ABox : elles permettent de décrire les individus à l'aide de concepts et des rôles. La déclaration des rôles et des concepts, ainsi que leurs paramètres sont déclarés au préalable dans la TBox. Pour la syntaxe concrète des assertions de la ABox, nous précédons tout simplement cet ensemble de prédicats du préfixe "*Assertions*". Pour les déclarations de la TBox, nous les décrivons comme un ensemble de prédicats avec le préfixe "*Declarations*" (cf. Code 8).

```
Assertions ::= 'Assertions' '{' {<Predicate>} '}'

Declarations ::= 'Declarations' '{' {<Predicate> '}'
```

Code 8: Syntaxe concrète pour les assertions

2.2.3.3. La représentation des actions

Nous choisissons une représentation d'action basée sur STRIPS, comme décrite dans la littérature, avec : un nom, un ensemble de préconditions, un ensemble de postconditions, et

des arguments. La syntaxe abstraite de cette structure est représentée par la Figure 29 qui est un fragment de la Figure 28.

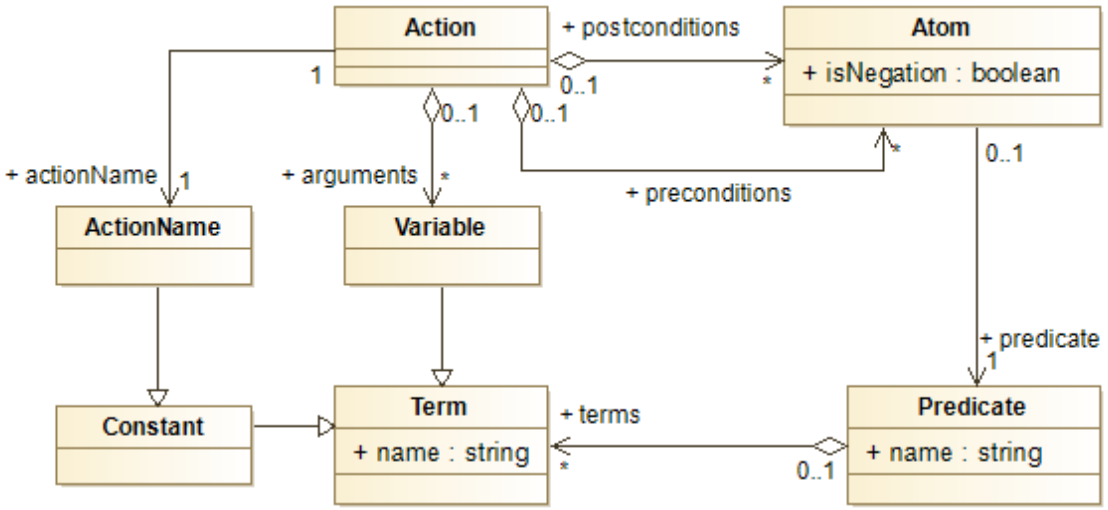


Figure 29: Syntaxe abstraite d'une action dans le présent travail.

Les postconditions sont une liste d'atomes, dont les atomes positifs représentent la *add-list*, et les atomes négatifs la *delete-list*. Pour décrire une action dans le langage que nous proposons, la syntaxe concrète en EBNF du Code 9 est utilisée pour décrire une action à la STRIPS avec ses préconditions, et ses postconditions .

```
Action ::= 'Action' <ActionName> '(' {<Variable>}')' '{'
     'Preconditions' '{' <Action.preconditions> '}' ','
     'Postconditions' '{' <Action.postconditions> '}'
'}'
```

Code 9: Syntaxe concrète d'une action

Par exemple pour l'action de chasser, elle peut être décrite par la syntaxe du Code 10.

```
Action hunt(var x, var z1) {
      Preconditions{at(x,z1)}, Postconditions{hasMeat(x)}
}
```

Code 10: Un exemple d'action décrite avec la syntaxe concrète du langage proposé

Nous choisissons de représenter les actions de manière déterministe à l'instar de STRIPS [Fikes & Nilsson, 1971], où les conséquences des actions sont certaines et apparaissent simultanément lorsque l'action est exécutée. La prise en compte des conséquences incertaines, conditionnelles et duratives des actions ne figure pas dans le cadre du présent travail.

2.2.3.4. Syntaxe des normes régulatrices

Les normes régulatrices sont basées sur la notion de norme régulatrice dans ADICO de [Crawford & Ostrom, 1995]. Dans le cadre du présent travail, nous omettons le *"O"* d'ADICO, c'est-à-dire les sanctions et les récompenses puisqu'elles figurent au-delà du sujet abordé. Une norme régulatrice selon la grammaire ADIC est présentée sur la Figure 30.

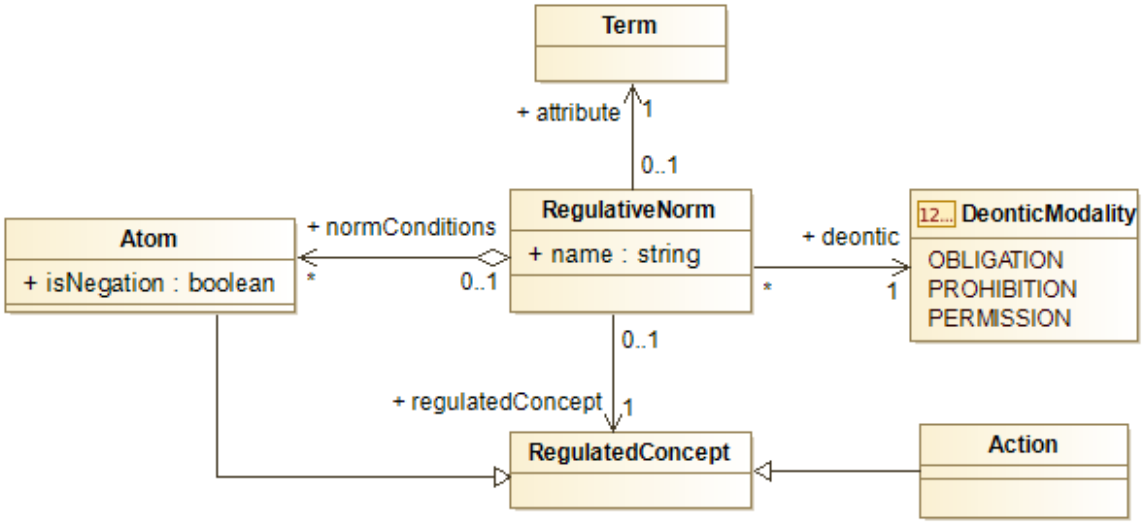


Figure 30: Syntaxe abstraite d'une norme régulatrice

Pour pouvoir les écrire dans le langage proposé, nous proposons la syntaxe concrète illustrée par le Code 11 avec la méta-syntaxe EBNF.

```
RegulativeNorm ::= ‘RegulativeNorm’ <RegulativeNorm.name> ‘(‘
      {<Variable>}
‘)’ ‘{‘
   [‘if’ ‘(‘ <RegulativeNorm.normConditions> ‘)’ ‘then’] <deonticModality>
   ‘(‘ ‘Attribute’ <attribute> ‘,’ <RegulatedConcept> ‘)’
‘}’
```

Code 11: Syntaxe concrète d’une norme régulatrice

Dans le cas où la norme n’a pas de condition d’applicabilité, nous pouvons omettre la partie “*if* ... *then*”, la norme devient de ce fait constamment applicable. Par exemple, pour décrire la norme qu’il est interdit pour un membre du village de pêcher dans une rivière sacrée, nous pouvons l’écrire comme présentée sur le Code 12.

```
RegulativeNorm fishingProhibition(var agent, var zone) {
      if (agent(member), sacred(zone))
      then prohibition(Attribute agent, fish(agent, zone))
}
```

Code 12: Un exemple de norme régulatrice interdisant la pêche dans une zone sacrée

2.2.3.5. Syntaxe des normes constitutives

Avec la notion de rôle et la notion de constante qui nomment les objets du système, nous synthétisons la syntaxe abstraite des normes constitutives à travers la Figure 31.

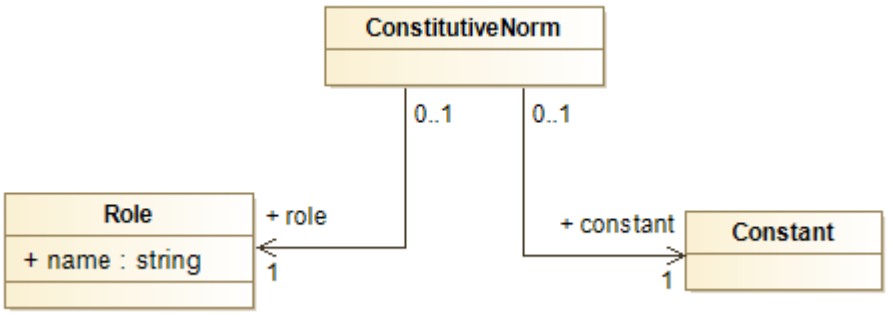


Figure 31: Syntaxe abstraite d'une norme constitutive

Pour exprimer une norme constitutive, nous reprenons le nom du rôle en tant que prédicat unaire (à un seul terme), et l'intégrons dans la syntaxe concrète des normes constitutives en tant que prédicat, comme le démontre la syntaxe concrète présentée dans le Code 13.

```
ConstitutiveNorm ::= <Role.name> '('<Constant>')'
```

Code 13: Syntaxe concrète d'une norme constitutive

Par exemple, pour désigner qu'une constante C1 joue le rôle de "sacrée", et que le rôle *babakoto*[14] de l'institution *biodiversity* compte comme un rôle *animal* de l'institution *exploitation*, elles peuvent être décrites à travers la syntaxe : *sacred(C1), animal(babakoto)*. Une notation pointée peut également être utilisée pour différencier les rôles de même nom venant d'institutions différentes : noter "*exploitation.animal*" au lieu de "*animal*".

2.2.3.6. Syntaxe abstraite des institutions

Une institution est faite d'un ensemble de normes régulatrices, de normes constitutives, d'assertions, de rôles et d'actions rattachées à ces institutions. Nous reprenons ces définitions à travers la Figure 32.

[14] Espèce de primates lémuriformes endémique à Madagascar

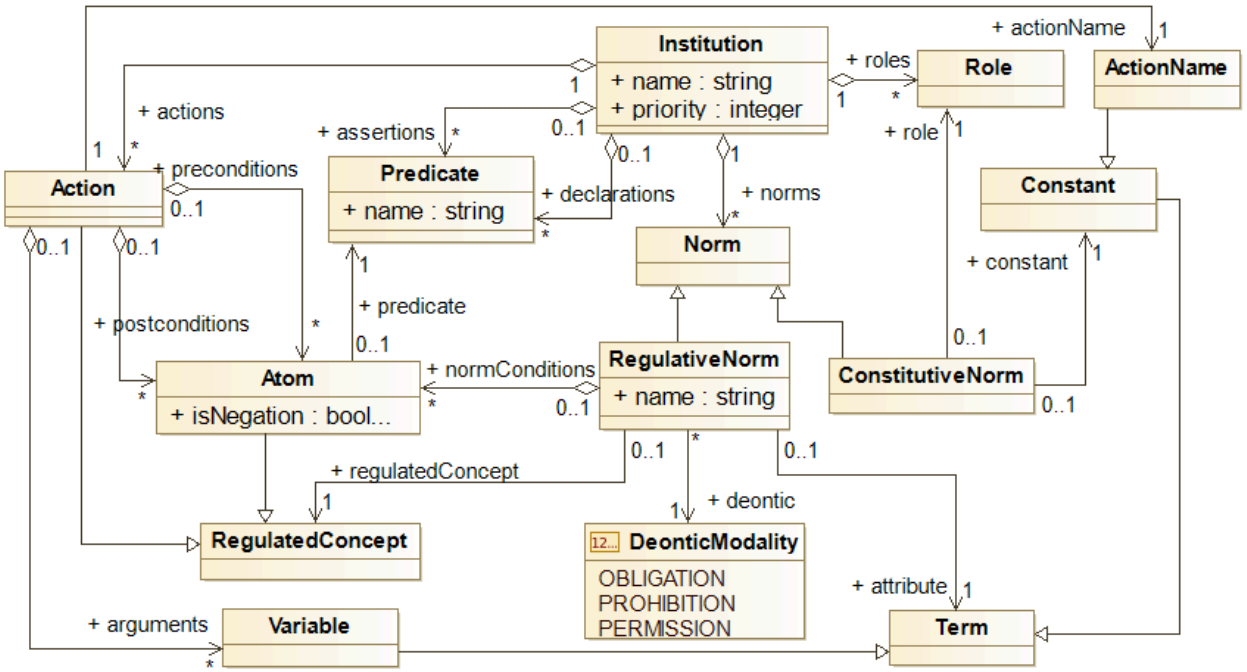


Figure 32: Syntaxe abstraite d'une institution

Poursuivant les travaux de [Aubert et al., 2010; Aubert & Müller, 2013; Raharivelo & Müller, 2018], nous ajoutons la notion de priorité à chaque institution: une priorité d'institution est un entier positif qui détermine l'importance de l'institution pour l'agent qui lui permettra plus tard de choisir quelles normes venant de quelle institution il peut se permettre de violer. L'algorithme de violation des normes basé sur la priorité de son institution est détaillé dans la section 2.8.Violation des normes.

2.2.3.7. Syntaxe concrète des institutions

Pour mettre à disposition de l'utilisateur la possibilité de décrire des institutions, la syntaxe concrète pour décrire une institution est décrite sur le Code 14 avec la méta-syntaxe EBNF.

```
Institution ::= ‘Institution’ <Institution.name> ‘{‘
      ‘Roles’ ‘:’ ‘{‘ {<Role>} ‘}’ ‘,’
      ‘Assertions’ ‘:’ ‘{‘ {<Predicate>} ‘}’ ‘,’
      ‘Declarations’ ‘:’ ‘{‘ {<Predicate>} ‘}’ ‘,’
      ‘Actions’ ‘:’ ‘{‘ {<Action>} ‘}’ ‘,’
      ‘Norms’  ‘:’ ‘{‘ {<Norm>} ‘}’ ‘,’
      ‘Priority’ ‘:’ +digit
}
```

Code 14: Syntaxe concrète d’une institution

Dans cette syntaxe, les normes constitutives seront représentées par des prédicats unaires, i.e. à 1 terme. Par exemple, pour décrire une institution spécifique avec cette syntaxe concrète, le Code 15 décrit une institution, d’une priorité maximale (10) avec :

- Le rôle de membre(*member*) du village;
- L’action de bouger d’une zone à une autre avec ses préconditions, et ses postconditions ;
- l’obligation de payer des taxes dans tout territoire sacré.

Pour cela, l’institution doit également avoir des assertions : être dans un endroit *P*, que *P* soit considéré comme un territoire sacré. Pour donner un exemple d’institution, nous avons un exemple d’institution sur le Code 15.

```
Institution village{
	Roles : { Role member, Role sacred, Role zone, Role agent},
	Declarations: {
		at(agent, zone), haveMeat(agent)
	},
	Actions : {
		Action hunt(var x, var z1) {
			preconditions {at(x, z1)},
			postconditions {hasMeat(x)}
		}
	},
	Norms : {
		RegulativeNorm noHunting(var x, var z) {
			if(member(x), at(x, z), sacred(z))
			then prohibition(Attribute x, hunt(x, z))
		}
	},
	priority : 10
}
```

Code 15: Exemple d'une description d'institution avec la syntaxe concrète du langage

2.2.4. Syntaxe des organisations

Les organisations (cf section 1.1.2.) en tant qu'instance des institutions sont représentées dans le présent travail avec un système prédicatif. À l'aide de normes constitutives, nous pouvons décrire dans une organisation qu'un certain objet compte comme un agent mobile, et qu'en tant que tel, il peut bouger:

- "Un objet λ compte comme un agent" ;
- "Un objet λ compte comme un objet mobile" (et acquiert donc l'action de bouger);
- "Une zone z_1 est la voisine d'une zone z_2";
- "Un objet λ se situe dans z_1".

Puis pour une autre institution villageoise, nous pouvons le lier à d'autres rôles tel que le rôle de pêcheur, par exemple, l'institution villageoise peut spécifier que :

- "Un objet λ compte comme un pêcheur" (et acquiert donc l'action de pêcher);
- "Il est interdit pour tout pêcheur de pêcher dans un village sans licence" où pêcheur et village sont des rôles;
- Si l'on a "λ compte comme un pêcheur" et "*V1* compte comme un village", alors nous pouvons déduire par inférence : "Il est interdit pour λ de pêcher dans *V1* sans licence"

À l'aide d'un sous-ensemble de la figure Figure 28, nous formalisons les composants d'une organisation et d'une institution dans la syntaxe abstraite de la Figure 33.

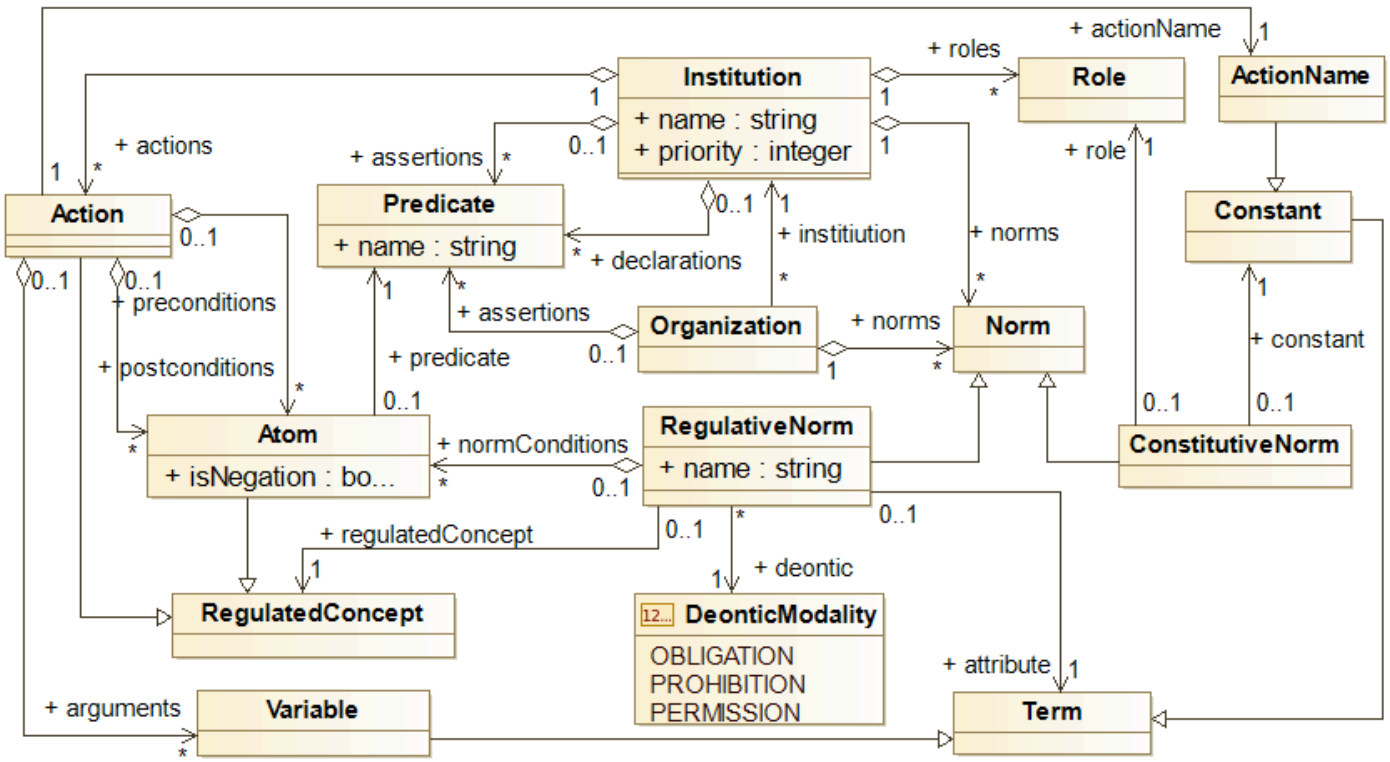


Figure 33: Syntaxe abstraite complète des organisations et des institutions

Pour décrire formellement une organisation, il nous suffit de référencer l'institution qu'elle instancie, ainsi que les normes constitutives de l'organisation permettant de décrire quel objet joue quel rôle, y compris les agents. La syntaxe concrète des organisations se présente à travers le Code 16 avec la meta-syntaxe EBNF.

```
Organization ::= 'Organization' '{'

        'Institution' ':' Institution ','

        'Assertions' ':' '{' {<Predicate>} '}' ','

        'Norms'  ':' '{' {<Norm>} '}'

'}'
```

Code 16 : Syntaxe concrète d'une organisation en EBNF

Par exemple, pour instancier un village, il suffit d'écrire le Code 17 pour pouvoir instancier l'institution village décrite précédemment. Les organisations peuvent ajouter des normes locales à l'organisation qui ne font pas partie des normes de l'institution.

```
const RAKOTO, const Z1, const Z2, const Z3,
Organization {
      Institution : village,
      Assertions : {
            agent(RAKOTO), member(RAKOTO),
            sacred(Z3), at(RAKOTO, Z1)
      }
      Norms : { }
}
```

Code 17 : Exemple de la syntaxe concrète d'une organisation.

2.2.5. Syntaxe d'un problème de planification normatif

Ayant défini les syntaxes des concepts pour prendre en compte les normes, nous pouvons à présent décrire la syntaxe concrète du problème de planification. Pour plus d'uniformité, les éléments classiques d'un problème de planification (objectifs, répertoire d'actions, et situation initiale), seront ajoutés dans une institution par défaut que nous nommons arbitrairement "global", de ce fait, un problème de planification normatif peut être spécifié à l'aide d'organisations seulement. Ces approches seront détaillées dans cette section pour le cas de la situation initiale, des objectifs, et du répertoire d'actions.

2.2.4.1. Syntaxe de la situation initiale

Dans la description du problème de planification classique, la situation initiale est composée de prédicats. Puisque dans nos contributions, le concept d'assertion permet de décrire des situations concrètes, nous proposons donc de spécifier cette situation initiale à travers les assertions de l'organisation globale et de son institution.

À l'instar du Code 18, nous pouvons par exemple en faire une situation initiale en la spécifiant comme une organisation de l'institution globale.

```
const RAKOTO, const Z1, const Z2, const Z3,
Organization {
      Institution : global,
      Assertions : {
            agent(RAKOTO), member(RAKOTO),
            sacred(Z3), at(RAKOTO, Z1)
      }
      Norms : { }
}
```

Code 18 : Exemple de spécification d'une situation initiale à travers l'organisation globale

2.2.5.1. Syntaxe de l'objectif

Pour pouvoir représenter un objectif, nous proposons de les définir en tant qu'obligation dans l'organisation globale que l'agent cherche à satisfaire. Par exemple, pour décrire que l'agent a pour objectif d'avoir de la viande (*haveMeat*) nous pouvons la décrire dans le fragment de l'organisation globale comme le présente le Code 19.

```
Organization {
      Institution : global, Assertions : { } ,
      Norms : { RegulativeNorm mustHaveMeat {
            obligation(Attribute SELF, haveMeat(SELF)
      }
}
```

Code 19: Syntaxe concrète d'un objectif à travers l'organisation globale

2.2.5.2. Syntaxe du répertoire d'actions

Pour plus d'uniformité, le répertoire d'actions initial sera représenté par l'ensemble des actions de l'institution globale. Le répertoire d'actions n'est donc plus fixé par le problème de planification, mais sera dynamique en fonction des organisations pertinentes. Pour décrire par exemple qu'il est possible de chasser selon le problème de planification, nous proposons de le décrire selon le Code 20 dans l'institution globale.

```
Institution global {
      Roles : { }, Assertions : { },
      Actions : {
            Action hunt(var x, var z1) {
                  preconditions {at(x, z1)},
                  postconditions {hasMeat(x)}
            }
      },
      Norms : { },
      Priority: 10
}
```

Code 20 : Exemple d'une spécification du répertoire d'actions initial

Après avoir formulé ce problème de planification normative, il faut décrire comment ces nouveaux concepts vont impacter sur le planificateur lui-même. Les extensions apportées au POP dans cette optique sont donc décrites dans les sections suivantes.

2.3. Extensions du POP

Cette section décrit la sémantique du problème de planification une fois que celle-ci est établie. Pour cela, de nouveaux concepts sont introduits dans les structures du POP classique : les états et les opérateurs.

2.3.1. Extension des états

Les états dans le POP classique sont définis par des plans à ordre partiel, dont on corrige les lacunes jusqu'à obtenir un plan exécutable qui accomplissent l'objectif, s'il y en a. Soit le plan à ordre partiel $\mathcal{P} = < ?, \wp, Cc, Ct, F >$ où :

- *S* est l'ensemble des situations du plan;
- *A* est l'ensemble des pas du plan;
- *Cc* est l'ensemble des contraintes de (non) codénotation;
- *Tc* est l'ensemble des contraintes temporelles;
- *F* est l'ensemble des lacunes du plan qui peuvent être soit des conditions ouvertes, soit des menaces.

Nous proposons d'étendre *F* avec de nouveaux types de lacunes normatives (en plus des conditions ouvertes, et des menaces, cf. 1.6.1.4. Les lacunes) pour pouvoir proposer de nouveaux opérateurs, permettant de résoudre ces nouvelles lacunes introduites dans [Ramarozaka et al., 2021].

Pour intégrer nos extensions dans l'état des POP (=un plan à ordre partiel), nous définissons une extension de celle-ci : un plan normatif à ordre partiel, qui peut contenir des lacunes, y compris des lacunes normatives, nous redéfinissons notamment deux méthodes :

- *getInitialeState()* : retourne le plan normatif à ordre partiel initial qui se construit de la même façon qu'un plan à ordre partiel classique avec deux pas fictifs, la situation initiale, la situation finale (conformément à 1.6.5.6.Satisfaction des objectifs);
- *getOptions(State)* : qui va retourner de nouveaux opérateurs pour résoudre les nouvelles lacunes normatives spécifiées;
- *isFinal(State)* : qui doit rendre le plan exécutable, mais aussi qui ne doit pas comporter de lacunes normatives.

Pour calculer l'ensemble des normes applicables à l'agent, nous n'avons qu'à déterminer quelles sont les organisations pertinentes dans le problème de planification, récupérer les normes de ces organisations, puis vérifier leurs conditions dans chaque situation du plan. Pour cela, nous donnons la définition du nouvel état dans le POP normatif.

Définition *(État dans le POP normatif) : dans le POP normatif, l'état encode un plan normatif à ordre partiel qui est un 5-uplet* $< \square, \wp, Cc, Ct, F, O' >$ *où :*

- $\square$ *désigne l'ensemble des situations;*
- $\wp$ *désigne l'ensemble des pas, c'est-à-dire des instances d'action;*
- *Cc désigne l'ensemble des contraintes de (non) codénotation;*
- *Ct désigne l'ensemble des contraintes temporelles qui définissent un ordre partiel sur* $\square \cup \wp$*;*
- *F désigne l'ensemble des lacunes du plan: l'ensemble des conditions ouvertes, des menaces du plan, et des lacunes normatives;*
- O' *désigne l'ensemble des organisations formulées dans le problème de planification.*

Pour illustrer les extensions apportées à l'état du POP classique, la Figure 34 retrace la structure du nouvel état dans le POP normatif, dont les différences avec un état classique ont été colorées en rouge pour être mises en exergue.

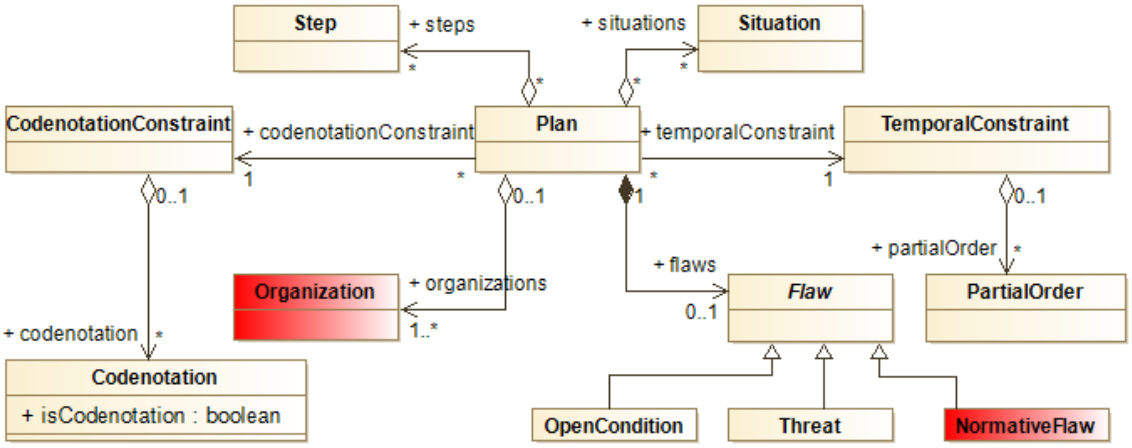


Figure 34: Nos extensions sur l'état classique du POP (en rouge)

Plusieurs processus doivent alors être mis en œuvre et seront décrits dans les prochaines sections:

1) Détecter les normes applicables;
2) Générer les lacunes normatives correspondantes en cas de violation des normes applicables;

3) Proposer des opérateurs pour résoudre les lacunes normatives ou bien des stratégies pour prendre en compte les normes.

2.3.2. Détection des normes applicables

Dans un état, il faut calculer pour chaque situation l'ensemble des normes applicables. Cet ensemble de normes applicables est l'ensemble des normes de toute organisation pertinente. Nous appelons "normes d'une organisation" l'ensemble des normes dans une organisation, et de l'institution dont elle est l'instance. Une organisation est dite "pertinente" si l'agent y joue un rôle selon le problème de planification.

Pour chaque problème de planification, nous pouvons construire la liste des organisations pertinentes grâce au problème de planification, parmi lesquelles on extrait les normes applicables. L'algorithme pour détecter les lacunes normatives se déroule comme suit :

1) Pour chaque situation du plan, nous allons récupérer l'ensemble des normes applicables, et vérifier si elles sont effectivement appliquées;
2) Selon la modalité déontique et le type de concept régulé, une nouvelle lacune normative sera créé :
    a) Si une action obligatoire n'est pas présente dans le plan alors que celle-ci est applicable, une obligation manquante sera générée;
    b) Si une action ou une condition interdite est présente dans une situation où elle est applicable, une interdiction manquante sera générée;
    c) Pour les autres types de normes et de concepts régulés, d'autres stratégies seront appliquées.

Selon leur modalité déontique (obligation, interdiction, ou permission) et le type du concept régulé (condition, ou action). Nous aurons donc à traiter six (06) formulations de normes possibles qui peuvent impacter sur le comportement engendré:

1) des conditions obligatoires;
2) des actions obligatoires;

3) des conditions interdites;
4) des actions interdites;
5) des conditions permises;
6) des actions permises.

Ces différentes formulations peuvent donner lieu à différentes lacunes normatives ou d'autres algorithmes qui vont indirectement impacter sur le processus de planification, si la norme est applicable et qu'elle n'est pas encore prise en compte par le plan.

2.3.3. Nouvelles lacunes et nouveaux opérateurs

Puisque les opérateurs du POP se basent sur la résolution de lacunes, nous introduisons les nouvelles lacunes relatives aux notions de normes : les lacunes normatives, ainsi que les nouvelles modifications du plan qui permettent de les résoudre [Ramarozaka et al, 2022].

2.3.3.1. **Lacune sur les actions obligatoires**

Pour une action obligatoire, celle-ci réclame que l'action correspondante soit nécessairement au moins exécutée une fois au cours du plan, sinon, une lacune normative est créée : une action obligatoire manquante.

Définition *(action obligatoire manquante)* : une action obligatoire manquante est une lacune normative qui apparaît si et seulement si les conditions d'applicabilité d'une obligation θ sont nécessairement vraies dans une situation S, et l'action obligatoire n'est pas présente dans le plan.

Pour pouvoir résoudre une action obligatoire manquante, de nouveaux opérateurs sont introduits. En posant S comme étant la situation où la lacune est applicable, nous pouvons la résoudre :

- *Par Adaptation:* on ajoute une instance de l'action obligatoire, notée ϕ, que l'on place juste avant la situation finale, avec les contraintes de codénotation adéquates tirées de son contexte d'applicabilité. Formellement, en posant comme postconditions de ϕ l'ensemble $q_1, q_2, \ldots, q_n$, et comme objets de l'action obligatoire l'ensemble $arg_1, arg_2, \ldots, arg_n$ les modifications se résument à :
  - $S = S \cup s^{\phi}_{before} \cup s^{\phi}_{after}$
  - $\wp = \wp \cup \phi$
  - $Cc = Cc \cup (q_1 \sim arg_1) \cup (q_2 \sim arg_2) \cup \ldots (q_n \sim arg_n)$ où $arg_1, arg_2, \ldots, arg_n$ sont les substitutions rendant la norme applicable;
  - $Ct = Ct \cup (S_i \prec s^{\phi}_{before}) \cup (s^{\phi}_{after} \prec S_f)$
- *Par Codénotation :* on ajoute de nouvelles contraintes de codénotation sur un des pas existants dans le plan pour que celle-ci soit identique à l'action obligatoire, si cela ne rend pas les contraintes de codénotation contradictoires. Soit q_i les postconditions du pas ϕ choisi, et arg_i les objets de l'action obligatoire. Les modifications sur le plan se résument donc à :

$$Cc = Cc \cup (q_1 \sim arg_1) \cup (q_2 \sim arg_2) \cup \ldots (q_n \sim arg_n)$$

- *Par Contournement :* ajouter un pas d, tel que $\exists q \in postconditions(d) \mid q \sim \neg c$ où c désigne une des conditions d'applicabilité de la norme; La modification du plan se résume aux formules suivantes :
  - $S = S \cup s^{d}_{before} \cup s^{d}_{after}$
  - $\wp = \wp \cup d$
  - $Cc = Cc \cup (q \sim \neg c)$
  - $Ct = Ct \cup (d \prec S)$

Pour implémenter l'opérateur de contournement : Soit n une norme applicable dans une situation *s*, et $conditions(n)$ les conditions d'applicabilité de *n*. Pour chaque condition d'applicabilité p dans $conditions(n)$, nous allons créer une condition ouverte $OC_{\neg p,\, s}$ qui requiert que $\neg p$ soit nécessairement vraie dans s. Formellement, pour une norme *n* applicable dans une situation *s* :

$$\forall p \in condition(n),\ \exists OC_p\ tel\ que\ \square(\neg p, s)$$

Cette condition ouverte sera ainsi prise en charge par les opérateurs classiques d'une condition ouverte, dont l'ajout potentiel d'un nouveau pas dans le plan.

2.3.3.2. Lacune sur les atomes obligatoires

Pour le cas d'une condition obligatoire, celle-ci est rajoutée dans la liste des objectifs de la planification. C'est-à-dire que l'agent doit accomplir la condition obligatoire au terme de la planification. Formellement :

- soit $\boldsymbol{\mathcal{G}}$ les objectifs du problème de planification;
- soit $\Delta vrai(p, s)$ le fait qu'un atome p soit nécessairement vrai dans une situation s quelconque du plan;
- pour une norme η de modalité déontique 'permission', nous noterons *normconditions(*η*)* ses conditions d'applicabilité, et *regulatedConcept(*η*)* son objectif est un atome. Nous pouvons donc appliquer la formule :

  $$\forall p \in normconditions(\eta) | \Delta\, vrai(p, s) \Rightarrow \forall q | q \in regulatedConcept(\eta),\ q \in \boldsymbol{\mathcal{G}}$$

2.3.3.3. Lacune sur les actions interdites

Pour une action interdite, présente entre une situation S_i où l'interdiction est applicable, et une situation S_j où l'interdiction n'est plus applicable, telle que $S_i < S_j$, une lacune normative sera créée : une action interdite.

Définition *(action interdite) :* une action interdite est une lacune d'un plan normatif $\mathscr{P}$ qui apparaît lorsqu'une interdiction ℕ interdise une action α et :

- Il existe une situation s_i où les conditions d'applicabilité ℕ sont nécessairement vraies;
- Il existe une situation s_j avec la contrainte temporelle $s_i \prec s_j$ où les conditions d'applicabilité de ℕ ne sont plus nécessairement vraies;
- L'action α interdite par ℕ se trouve dans l'intervalle $[s_i, s_j]$, tel que $s_i \prec \alpha$ et $\alpha \prec s_j$.

Pour résoudre cette lacune au sein du POP, de nouveaux opérateurs sont introduits. L'agent peut donc les résoudre :

- *Par Promotion :* extirper l'action interdite en la plaçant avant s_i si cela ne rend pas les contraintes temporelles contradictoire, i.e. on ajoute la modification suivante

$$Ct = Ct \cup (s^{\alpha}_{after} \prec s_i)$$

- *Par Démotion* : extirper l'action interdite en la plaçant après s_j, si cela ne rend pas les contraintes temporelles contradictoires, i.e. on va ajouter la modification suivante

$$Ct = Ct \cup (s_j \prec s^{\alpha}_{before})$$

- *Par Contournement* : ajouter ou choisir une action du plan qui rend nécessairement vrai la négation d'une des conditions d'applicabilité de la norme, exactement comme pour le contournement des obligations.

2.3.3.4. Lacunes sur les atomes interdits

Définition *(Atome interdit) :* Un atome interdit est une lacune d'un plan normatif $\mathcal{P}$, qui apparaît lorsque les conditions d'applicabilité de l'interdiction sont nécessairement vraies entre deux situations s_i et s_j et l'atome interdit p est vrai dans une situation s, tel que $s_i \prec s$ et $s \prec s_j$.

Pour résoudre un atome interdit présent, c'est-à-dire qu'un certain état du monde existe alors qu'il est actuellement interdit, on a les opérateurs suivants:

a) *démotion* : extirper l'atome interdit en plaçant son pas établisseur ξ après s_j, si cela ne rend pas les contraintes temporelles contradictoires, i.e. on va ajouter la modification suivante

$$Ct = Ct \cup (s_j \prec s^{\xi}_{before})$$

b) *promotion :* extirper l'atome interdit en plaçant son pas établisseur ξ avant s_i, si les contraintes temporelles n'en deviennent pas contradictoires. Autrement, dit on va ajouter la modification suivante

$$Ct = Ct \cup (s^{\xi}_{after} \prec s_i)$$

c) *contournement :* exactement comme la stratégie de contournement présentée dans la section précédente sur les actions obligatoires: on cherchera à rendre nécessairement vraie la négation de l'une des conditions d'applicabilité de la norme.

2.3.4. Autres algorithmes de prise en compte des normes

2.3.4.1. Atome obligatoire

Dans le problème de planification, les objectifs sont composés d'atomes que l'agent souhaite accomplir et qui sont représentés par les obligations de l'institution globale. Nous considérons qu'un atome obligatoire pris en compte se transforme donc en objectif.

Nous simplifions donc que les atomes obligatoires resteront tant que l'agent est membre de son organisation. Une proposition pour représenter l'aspect spatio-temporel des normes est notamment proposée dans [Raharivelo & Müller, 2018], dont l'intégration dans notre planificateur peut faire l'objet de travaux futurs.

2.3.4.2. Action permise

Pour le cas des actions permises, nous en déduirons que pour adopter la norme, il suffit d'ajouter les conditions d'applicabilité de la norme aux préconditions de l'action en question : nous introduisons donc, parmi les conditions mécaniques d'une action, ses conditions sociales qui décrivent quand est-ce qu'elle doit être permise. Formellement,

- Soit une norme η qui rend une action α permise.
- Si : $\exists s \in$ [illegible], $\forall c \in normconditions(\eta) \mid \square(c, s)$

alors: $preconditions(\alpha) = preconditons(a) \cup normconditions(\eta)$

Cette extension des préconditions mécaniques de l'action permise se fait lors du *getOptions*, puisque c'est au moment de résoudre une lacune que les actions sont utilisées : si elle est applicable dans la situation concernée par la lacune, l'injection des conditions d'applicabilité se fait, sinon, l'injection n'est pas effectuée. Le planificateur va par la suite chercher à rendre nécessairement vraies ces nouvelles préconditions, ce qui va alors requérir plus d'opérateurs par la suite.

2.3.4.3. Atome permis

Pour le cas des conditions permises, nous considérons que toute condition (=atome) est permise en toute circonstance en l'absence d'une permission explicite. Par contre, quand une condition est permise sous certaines conditions par une norme, elle est automatiquement interdite dans toute situation qui ne satisfait pas ses conditions d'applicabilité. Les permissions sont donc converties en interdictions.

Pour une permission η qui régule un atome p, et N l'ensemble des normes régulatrices d'une institution donnée, la conversion est décrite par la formule :

$$\forall c \in normconditions(\eta) | \Delta\, vrai(p, s) \Rightarrow \forall c,\ Prohibition(\neg c \rightarrow p) \in N$$

Où $Prohibition(\neg c \rightarrow p)$désigne une interdiction dont les conditions d'applicabilité sont décrites par $\neg c$, et a pour objectif l'atome p.

2.4. Violation des normes

Pour la violation des normes, nous reprenons les concepts de [Aubert et al., 2010] où chaque institution possède un niveau de priorité pour l'agent. Lorsque l'agent ne trouve pas de plan pour prendre en compte les normes, il commence par violer les normes de l'institution le moins de priorité.

Pour cela, nous remontons à la situation où la norme est applicable et la prise en compte de la norme, en ignorant toute lacune normative engendrée par les normes de l'institution. Ceci suppose donc que :

1) L'agent commence par chercher un plan qui se conforme aux normes;
2) C'est seulement en cas d'échec que l'agent viole les normes;
3) L'agent replanifie donc avec une liste d'institutions à ignorer, à commencer par l'institution avec la plus faible importance;
4) Lorsque la planification échoue à nouveau, une nouvelle institution est ajoutée à la liste des institutions à ignorer;
5) La planification s'arrête lorsqu'un plan solution est trouvé, ou quand toutes les institutions sont dans la liste des institutions à ignorer et aucun plan solution n'est trouvé.

En d'autres termes, nous considérons que l'agent commence par chercher un plan qui prend en compte toutes les normes avant de considérer un plan qui viole les normes et qui minimise les violations en se basant sur la priorité de chaque institution. Pour preuve de faisabilité, une implémentation de nos contributions est présentée dans la section 3.

3. Implémentation

Pour pouvoir mettre en avant la preuve de faisabilité de nos contributions, cette section vise à démontrer avec des exemples que 1) le planificateur traite en effet des normes : que ce soit les obligations, les interdictions, et les permissions; que ce soit sur des conditions (=atomes), ou des actions, et 2) le planificateur produit un comportement adapté et explicable qui prend en compte ces normes. Pour pouvoir démontrer que le planificateur prend en compte les normes, une implémentation du planificateur est présenté, suivi d'un problème de planification normatif comme preuve de faisabilité nommé IRIELA[15] est présenté dans cette section.

3.1. Implémentation du planificateur

Pour démontrer comment on peut implémenter cette théorie pratiquement, nous choisissons d'implémenter notre générateur de plans d'action normatif en utilisant le langage de programmation orienté-objet Java. Cette implémentation est disponible sur le lien dépôt GitHub suivant : https://github.com/tokyramarozaka/mimosa-iriela-extension.

Pour cette implémentation, nous mettons en œuvre les contributions mentionnées, avec le langage Java 17 sous un environnement de développement IntelliJ IDEA 2022. 1. 2, Édition Ultimate avec les bibliothèques suivantes :

- *log4j2* pour afficher les traces d'exécution du programme de manière textuelle;
- *Grep Console* pour mettre en exergue à l'aide d'une coloration les traces d'exécution importantes;
- *Graphviz* pour construire automatiquement des graphes représentant le parcours de l'espace de recherche,
- Et enfin, *JUnit5* pour les tests unitaires pour un développement dirigé par les tests, en séparant les différents scénarios.

[15] IRIELA résulte de la concaténation de deux mots Malgaches "irina" et "ela" signifiant respectivement "désiré" et "longtemps". Désigne quelque chose qu'on a désiré depuis longtemps (comme cette thèse).

Le reste de cette sous-section détaille: 1) la représentation de l'espace de recherche et l'algorithme de recherche choisi pour le parcourir, 2) les restitutions visuelles utilisées pour présenter les plans obtenus par le planificateur.

3.1.1. Représentation de l'espace d'états

Pour pouvoir représenter l'espace de recherche en forme d'arbre, et suivre son expansion en résolvant les lacunes (potentiellement normatives), nous mettons en place une hiérarchie d'états pour comprendre: i) quel état succède à quel état? ii) Quel opérateur a été utilisé pour passer à l'état suivant ?, iii) récupérer la longueur du parcours effectué pour atteindre l'état actuel (afin de l'intégrer dans le calcul heuristique).

Pour représenter cet arborescence, une structure de données est créé : le *ProblemState*, représentée par la Figure 35, il se compose de :

1) L'état actuel lui même;
2) Son état parent qui est aussi représentée par un *ProblemState*;
3) Une fonction d'évaluation f, obtenue par la somme de la longueur de la recherche g (somme des evaluateOperator) et de la distance heuristique h qui est représentée par le nombre de lacunes (evaluateState);
4) L'opérateur qui a été appliqué à l'état parent pour obtenir l'état actuel.

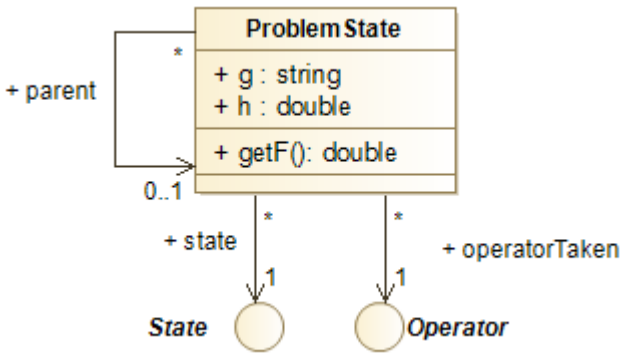


Figure 35: Description du *ProblemState* qui représente l'arborescence des états dans l'espace de recherche

La principale utilité de cette structure qu'est le *ProblemState* est de récupérer les états précédents et tous les opérateurs appliqués pour passer de l'état initial à l'état final. Pour cela, il part d'un *ProblemState* contenant l'état solution, et remonte à son parent, et récupère les opérateurs appliqués dans le bon ordre[16]. Pour parcourir cet espace de recherche dans cette implémentation, nous utilisons un algorithme classique qu'est A* [Likhachev et al., 2005].

3.1.2. Restitution visuelle

Pour pouvoir vérifier le résultat de la recherche dans l'espace d'états, et du plan obtenu, deux artéfacts de visualisation sont utilisés :

1) Les traces d'exécution de la planification grâce à log4j2;
2) Les graphes de visualisation de chaque état, mais aussi du parcours de l'espace d'états ainsi que les lacunes résolues grâce à Graphviz.

Pour offrir une visualisation textuelle de l'espace de recherche, nous traçons de manière textuelle l'exécution des processus suivants, en utilisant la coloration de Grep Console pour mettre en avant :

1) Le *getInitialState* pour décrire l'état initial, point de départ de la recherche;
2) Chaque *getOptions* pour liste l'ensemble des opérateurs applicables à un état;
3) Chaque *apply*, c'est-à-dire chaque application d'opérateur, ainsi que les états suivants obtenus;
4) Chaque *isValid* pour comprendre pourquoi certains états ont été exclus de l'espace de recherche;
5) Chaque *evaluateState* (le nombre de lacunes) et chaque *evaluateOperator*(la longueur du parcours), pour avoir une estimation de la fonction heuristique de l'état obtenu après chaque *apply*.
6) Le plan à ordre partiel solution obtenue (s'il y en a).

[16] Le bon ordre peut varier si l'on utilise un chainage avant VS. un chaînage arrière.

Par exemple, les Figure 36 et Figure 37 retracent les premières étapes d'un exemple de planification, avec notamment l'état initial, le premier *getOptions*, ainsi que le premier *apply*.

```
[DEBUG] aStar_planning.pop.PopPlanningProblem - _________GET OPTIONS_________
[INFO ] aStar_planning.pop.PopPlanningProblem - CURRENT STATE : PLAN
--SITUATIONS
    [Situation1, Situation2]
--STEPS
    [initial():3, final(X):2]
--CODENOTATIONS :
    CodenotationConstraints(codenotations=[])
--TEMPORAL CONSTRAINTS :
    TemporalConstraints(partialOrders=[initial():3 < Situation1, Situation1 < Situation2, Situation2 < final(X):2])
--FLAWS :
    [(OC): haveWood(X) IN Situation2]
-- ORGANIZATIONS :
    [globalOrg, villageOrg, householdOrg]
[INFO ] aStar_planning.pop.PopPlanningProblem -
______OPTIONS ARE______
[INFO ] aStar_planning.pop.PopPlanningProblem - --> OPTION #1 : RESOLVING (OC): haveWood(X) IN Situation2
Added :
    cut(X,Z):42
Added :
    [Situation3, Situation4]
Added :
    CodenotationConstraints(codenotations=[C42::X == C2::X])
Added :
    TemporalConstraints(partialOrders=[Situation3 < cut(X,Z):42, cut(X,Z):42 < Situation4, Situation4 < Situation2, Situation1 < Situation3
```

Figure 36: Illustration du premier getOptions d'un problème de planification qui consiste à trouver du bois (haveWood)

```
[INFO ] aStar_planning.pop.PopPlanningProblem - ___APPLYING___ RESOLVING (OC): haveWood(X) IN Situation2
Added :
    cut(X,Z):49
Added :
    [Situation5, Situation6]
Added :
    CodenotationConstraints(codenotations=[C49::X == C2::X])
Added :
    TemporalConstraints(partialOrders=[Situation5 < cut(X,Z):49, cut(X,Z):49 < Situation6, Situation6 < Situation2, Situation1 < Situation5
[INFO ] aStar_planning.pop.PopPlanningProblem -
___GOT___ PLAN
--SITUATIONS
    [Situation1, Situation2, Situation5, Situation6]
--STEPS
    [initial():3, final(X):2, cut(X,Z):49]
--CODENOTATIONS :
    CodenotationConstraints(codenotations=[C49::X == C2::X])
--TEMPORAL CONSTRAINTS :
    TemporalConstraints(partialOrders=[initial():3 < Situation1, Situation1 < Situation2, Situation2 < final(X):2, Situation5 < cut(X,Z):49
--FLAWS :
    [(OC): haveExploitationLicense(X) IN Situation5, (OC): containsTrees(Z) IN Situation5]
-- ORGANIZATIONS :
    [globalOrg, villageOrg, householdOrg]
[INFO ] aStar_planning.pop.PopPlanningProblem - HEURISTIC : 2
[INFO ] aStar_planning.pop.PopPlanningProblem - IS VALID : true
```

Figure 37: Illustration du premier apply dans le problème de planification généré par log4j2

Une autre façon de pouvoir restituer visuellement l'espace de recherche est de le représenter directement à travers un graphe l'expansion de l'espace de recherche rendu automatiquement par Graphviz. En raison de la très grande taille de l'espace de recherche, des simplifications ou une visualisation plus progressive sont nécessaires pour pouvoir le visualiser dans les prochaines sections.

3.2. Spécification du problème de planification IRIELA

Pour pouvoir démontrer comment le planificateur prend en compte les normes, le problème de planification IRIELA est représenté dans la Figure 38. Nous allons spécifier la structure de ses composants en utilisant le langage proposé et observer les comportements obtenus par le planificateur et les interpréter.

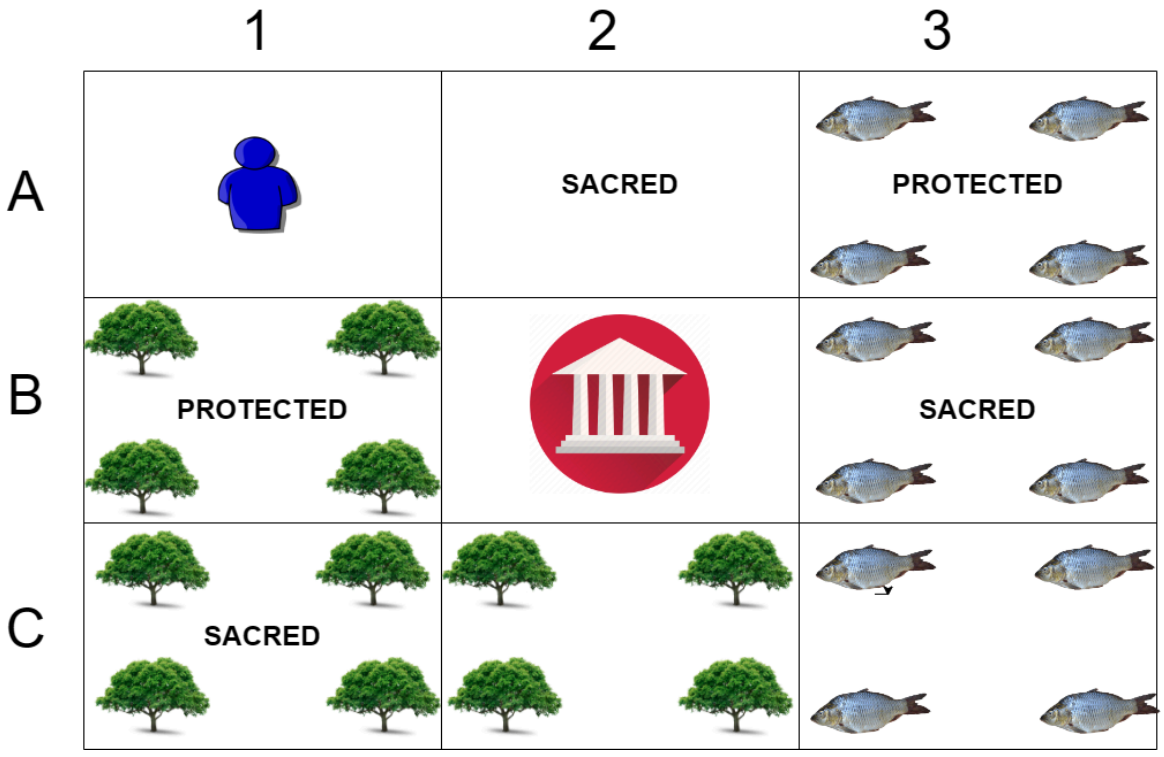


Figure 38: Synthétisation schématique du problème de planification IRIELA

Dans ce problème de planification, l'agent se trouve dans un monde sous la forme d'une grille 3x3 nommés : A1, A2, A3, B1, B2, B3, C1, C2, et C3. La situation initiale du problème de planification consiste à avoir :

- L'agent situé en A1;
- Des poissons dans A3, B3, et C3;
- Des forêts dans B1, C1, et C2;
- Un bureau communal en B2.

Pour pouvoir simplifier la formulation du problème de planification normatif, nous allons tout d'abord en décrire ses composants, i.e. ses institutions, et ses organisations avec leur représentation en logique de prédicats du premier ordre.

3.2.1. Les institutions du problème

En se basant sur les notions de normes régulatrices, de rôles et d'actions, nous pouvons à présent décrire les institutions du problème de planification de manière exhaustive, avec une syntaxe appropriée. Dans le cas d'IRIELA, nous distinguerons les institutions :

- *global :* qui va décrire le monde, d'un point de vue topologique et comment on peut s'y déplacer;
- *household :* qui va décrire le point de vue d'un foyer qui doit survivre, et qui interprète le monde décrit par *global* comme étant des ressources pour sa survie*;*
- *village* : qui va décrire un système de régulation sur les activités des *household* pour respecter certaines coutumes ou pour pérenniser certaines ressources.

Par défaut, les actions disponibles à l'agent d'entrée de jeu seront mises dans l'institution globale que nous décrivons avec la syntaxe concrète décrite sur le Code 21. Selon le scénario de test, nous ajoutons l'obligation d'avoir du poisson (haveFish) ou du bois (haveWood) à titre d'objectif.

```
Institution global{
      Roles : { Role movingObject, Role zone, Role agent },
      Declarations : {
            located(agent, zone), areAdjacents(zone, zone)
      },
      Actions : {
            Action move(var X, var Z1, var Z2){
                  Preconditions {located(X, Z1),areAdjacents(Z1, Z2)},
                  Postconditions {not located(X, Z1), located(X, Z2)}
            }
      },
      Norms : { },
      Priority : 10
}
```

Code 21 : description de l'institution global d'IRIELA

L'institution *household* écrit sur le Code 22 représente le foyer qui a besoin de se nourrir et d'avoir du bois que le rôle de *provider* doit fournir.

```
Institution household {
      Roles : { Role provider, Role hasFish, Role haveFishingNet,
               Role haveWood, Role haveFish }
      Declarations : { },
      Actions : {
            Action fish(var X, var Z){
                  Preconditions {
                        global.located(X,Z), hasFish(Z)
                  },
                  Postconditions {
                        not hasFish(Z), haveFish(X)
                  }
            },
            Action cut(var X, var Z){
                  Preconditions{global.located(X, Z), hasTrees(Z) },
                  Postconditions{ not hasTrees(Z), haveWood(X) }
```

```
            }
      },
      Norms : {
            RegulativeNorm mustHaveFish(var X){
                  if(household.provider(X))
                  then obligation(attribute X,haveFish(X))
            },
            RegulativeNorm mustHaveWood(var X){
                  if(household.provider(X))
                  then obligation(attribute X, haveWood(X))
            }
      }
      Priority: 9.5
}
```

Code 22 : Description de l'institut household

L'institution villageoise, ou *village* décrite dans le Code 23 se charge de définir l'ensemble des territoires en relevant les forêts, les terres, et les bureaux communaux pour avoir les autorisations d'exploitation. Elle peut également définir quelles zones sont sacrées, protégées, quelles zones comprennent des arbres ou des poissons, et quelles zones n'en comportent plus.

```
Institution village{
      Roles : {
            Role sacred, Role protected, Role member, Role office
            Role haveFishingLicense, Role haveExploitaitonLicense,
            Role reportedFish
      },
      Declarations : {  },
      Actions :{
            Action reportFish(var X){
               Preconditions{haveFish(X)}, Postconditions{reportedFish(X)}
            },
            Action getFishingLicense(var X, var Z){
               Preconditions {member(X), global.located(X, Z), office(Z)},
               Postconditions{ haveFishingLicense(X)}
            },
            Action getExploitationLicense(var X, var Z){
               Preconditions {member(X), global.located(X, Z), office(Z)},
               Postconditions{haveExploitationLicense(X)}
            },
            Action getFishingNet(var X){
               Preconditions{ }, Postconditions{haveFishingNet(X)}
            }
      },
      Norms : {
            RegulativeNorm mandatoryFishReport(var X) {
               if (member(X), haveWood(X))
                  then obligatory(attribute X, reportFish(X))
            },
            RegulativeNorm locatedInSacred(var X, var Z){
               if (member(X), sacred(Z))
                  then prohibition(attribute X, global.located(X,Z))
            },
            RegulativeNorm fishWithoutLicense(var X, var Z){
               if (not haveFishingLicense(X), member(X), protected(Z),
                  global.located(X,Z))
                  then prohibition(attribute X, household.fish(X, Z))
            },
            RegulativeNorm cutWithLicense(var X, var Z){
               if (haveExploitationLicense(X), protected(Z))
                  then permission(attribute X, cut(X,Z))
            },
            RegulativeNorm haveFishingNetWithLicense(var X){
               if (member(X), haveFishingLicense(X))
                 then permission(attribute X, household.haveFishingNet(X))
            }
      },
      Priority : 7
}
```

Code 23: Représentation de l'institution village

3.2.2. Les organisations du problème

Lors de l'initialisation du problème de planification IRIELA, nous allons affecter les rôles suivants:

- L'agent compte comme un objet capable de se déplacer (*global*.movingObject), un membre du village (*village.member*), et un fournisseur pour son foyer (*household.provider*);
- Un ensemble de constantes (A1, A2, A3, B1, B2, B3, …) comptent comme des zones (*global.zone*);
- Du point de vue d'un village, une zone compte comme étant un des rôles : *village.land, village.forest, village.river, village. sacred ou village.protected*;

Soit la constante SELF, qui décrit l'agent lui même dans l'organisation globale, avec la syntaxe *const SELF.*. Nous pouvons décrire les rôles de l'agent avec la syntaxe suivante:

```
global.movingObject(SELF),

village.member(SELF),

household.provider(SELF)
```

De ces organisations, la liste des normes applicables à l'agent mis en place sont listées sur le Tableau 6, et dont les codes exhaustifs sont présentés en Annexe 4.

Tableau 6: L'ensemble des normes applicables pour un agent X.

Déontique	**Condition(s)**	**Objectif**
Obligation	provider(X)	haveWood(X)
Obligation	haveFish(X)	reportFish(X)
Interdiction	member(X), sacred(X, Z)	global.located(X, Z)
Interdiction	member(X), protected(Z), ¬haveFishingLicense(X)	household.fish(X, Z)
Permission	haveExploitationLicense(X)	cut(X, Z)
Permission	member(X), haveFishingLicense(X)	hasFishingNet(X)

3.3. Exploration des résultats

Puisque l'algorithme pour la prise en compte des normes ainsi qu'une preuve de faisabilité sont fournis, il nous faut à présent analyser les comportements engendrés dans celle-ci pour laisser transparaître "pourquoi" l'agent a pris telle décision plutôt qu'une autre. Nous allons donc analyser à chaque instance, deux scénarios : un scénario sans les normes, et un scénario avec les normes.

3.3.1. Comportements sous obligations

3.3.1.1. Atomes obligatoires

Pour le cas d'une condition obligatoire, si nous donnons à un agent dépourvu d'objectifs l'obligation de trouver du poisson, de par son rôle de chef de foyer (*household.provider*), l'agent va engendrer le plan de la Figure 39. Ceci est notamment dû au fait qu'avoir à du poisson à manger devient un objectif pour l'agent à partir du moment où celui-ci est applicable.

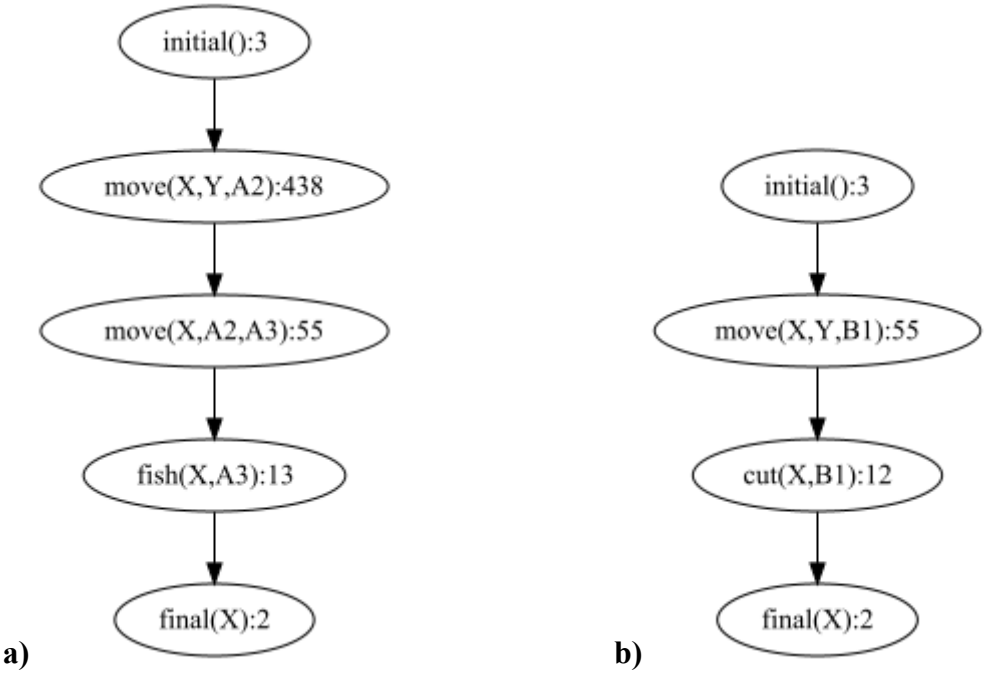


Figure 39: Plans obtenus, sans objectifs et avec uniquement l'obligation : a) d'avoir du poisson, et b) d'avoir du bois

En analysant les traces d'exécution textuelles, nous remarquons l'existence de conditions ouvertes pour les deux cas, par exemple, dans la Figure 40, nous pouvons constater que pour l'obligation de trouver du poisson dans l'institution household, engendre la condition ouverte *haveFish*.

```
[WARN ] aStar_planning.pop.PopPlanningProblem - _________GET OPTIONS_________
[INFO ] aStar_planning.pop.PopPlanningProblem - CURRENT STATE : PLAN #1
 - SITUATIONS : [Situation1, Situation2]
 - STEPS : [initial():3, final(X):2]
 - CODENOTATIONS :
    NONE
 - TEMPORAL CONSTRAINTS :
    initial():3 < Situation1,
    Situation1 < Situation2,
    Situation2 < final(X):2
 - FLAWS :
    (OC): haveFish(X) IN Situation2
-- ORGANIZATIONS :
    [globalOrg, householdOrg]
[INFO ] aStar_planning.pop.PopPlanningProblem - ______OPTIONS ARE______
[WARN ] aStar_planning.pop.PopPlanningProblem - --> OPTION #a TO RESOLVE (OC): haveFish(X) IN
  Situation2
Add step: fish(X,Z):10
Add situations: [Situation3, Situation4]
Add cc:
    [C10::X == C2::X]
Add tc:
    Situation3 < fish(X,Z):10,
    fish(X,Z):10 < Situation4,
    Situation4 < Situation2,
    Situation1 < Situation3
```

Figure 40: Trace d'exécution du fait d'avoir comme obligation d'avoir du poisson

Ce résultat démontre que les propositions obligatoires ne vont pas directement affecter le plan par des opérateurs, mais indirectement à travers leur conversion en objectifs. Sans les normes, les deux plans de la Figure 39 sont également logiques puisque :

- A3 est l'endroit le plus proche ayant du poisson en partant de A1;
- B1 est l'endroit le plus proche ayant du bois en partant de A1.

Nous nous servirons de ces deux plans obtenus comme référentiel par rapport aux autres plans obtenus en ajoutant d'autres types de norme.

3.3.1.2. Actions obligatoires

Pour démontrer le cas des actions obligatoires, nous appliquons en même temps deux normes de modalité déontique obligation illustrées par le Tableau 7.

Tableau 7: Liste des normes appliquées avec une action obligatoire et un atome obligatoire

Déontique	Condition(s)	Objectif
Obligation	provider(X)	haveWood(X)
Obligation	haveFish(X)	reportFish(X)

Pour illustrer le comportement produit en retour en un vrai plan à ordre partiel sans simplifier les contraintes temporelles par transitivité, la Figure 41 permet une visualisation complète de toutes les contraintes temporelles du plan à gauche, et une version simplifiée.

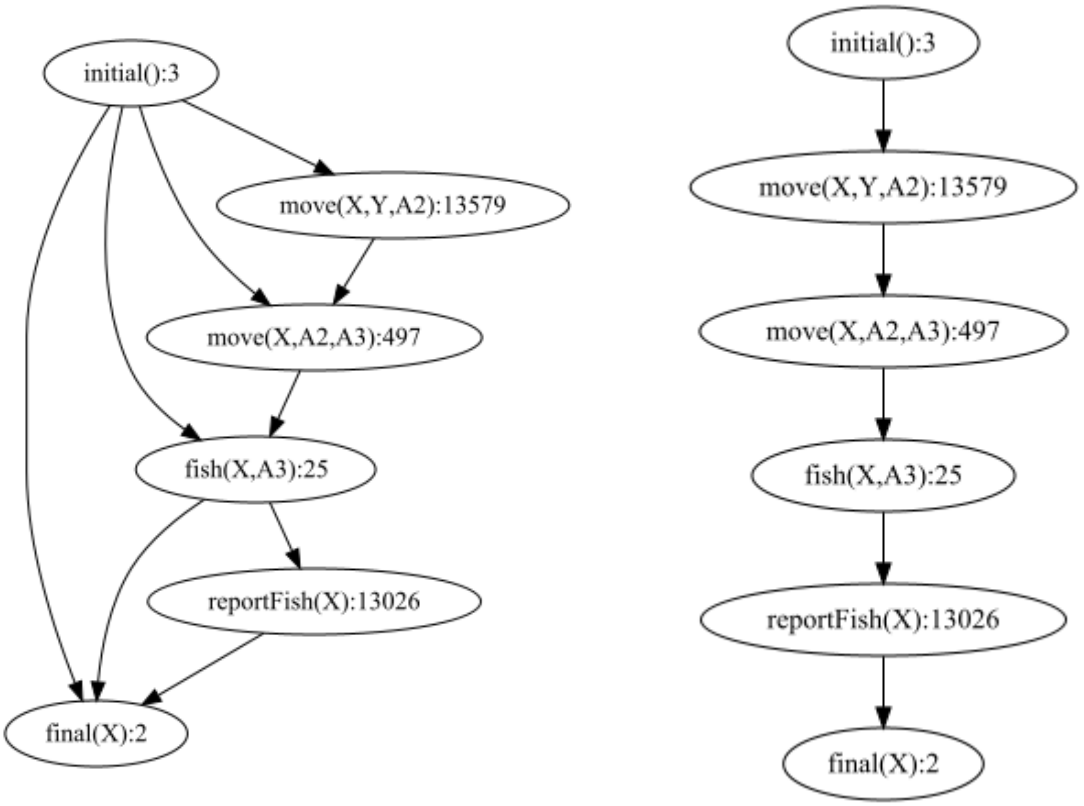


Figure 41 : Représentation simplifiée du plan obtenu avec l'obligation de trouver du poisson et de déclarer ses prises avec l'action *reportFish*.

Comparé au plan précédent où l'agent devait seulement trouver du poisson (Figure 39.a), nous pouvons constater que le comportement produit a ajouté la nouvelle action *reportFish* à son plan après avoir péché.

Pour comprendre pourquoi ce comportement est engendré, nous allons analyser les différentes options (=getOptions) pour comprendre comment les normes ont créé de nouvelles options dans l'espace de recherche avec les lacunes normatives (cf. Figure 42, dans la section FLAWS).

```
[WARN ] aStar_planning.pop.PopPlanningProblem - ___APPLY___ TO RESOLVE (OC): haveFish(X) IN
 Situation2
Add step: fish(X,Z):25
Add situations: [Situation5, Situation6]
Add cc:
    [C25::X == C2::X]
Add tc:
    Situation5 < fish(X,Z):25,
    fish(X,Z):25 < Situation6,
    Situation6 < Situation2,
    Situation1 < Situation5
[INFO ] aStar_planning.pop.PopPlanningProblem - ___GOT___ PLAN #2
 - SITUATIONS : [Situation1, Situation2, Situation5, Situation6]
 - STEPS : [initial():3, final(X):2, fish(X,Z):25]
 - CODENOTATIONS :
    C25::X == C2::X
 - TEMPORAL CONSTRAINTS :
    initial():3 < Situation1,
    Situation1 < Situation2,
    Situation2 < final(X):2,
    Situation5 < fish(X,Z):25,
    fish(X,Z):25 < Situation6,
    Situation6 < Situation2,
    Situation1 < Situation5
 - FLAWS :
    (OBLIGATION) : [member(X) , haveFish(X)] -> reportFish(X) IN Situation6
    (OBLIGATION) : [member(X) , haveFish(X)] -> reportFish(X) IN Situation2
    (OC): containsFishes(Z) IN Situation5
-- ORGANIZATIONS :
    [globalOrg, villageOrg, householdOrg]
[INFO ] aStar_planning.pop.PopPlanningProblem - HEURISTIC : 3
[INFO ] aStar_planning.pop.PopPlanningProblem - IS VALID : true
```

Figure 42: Première apparition de la lacune normative qu'il faut déclarer ses prises après avoir pêché.

La résolution de cette lacune peut se faire en ajoutant l'action obligatoire *reportFish()* au plan, comme le démontre la Figure 43.

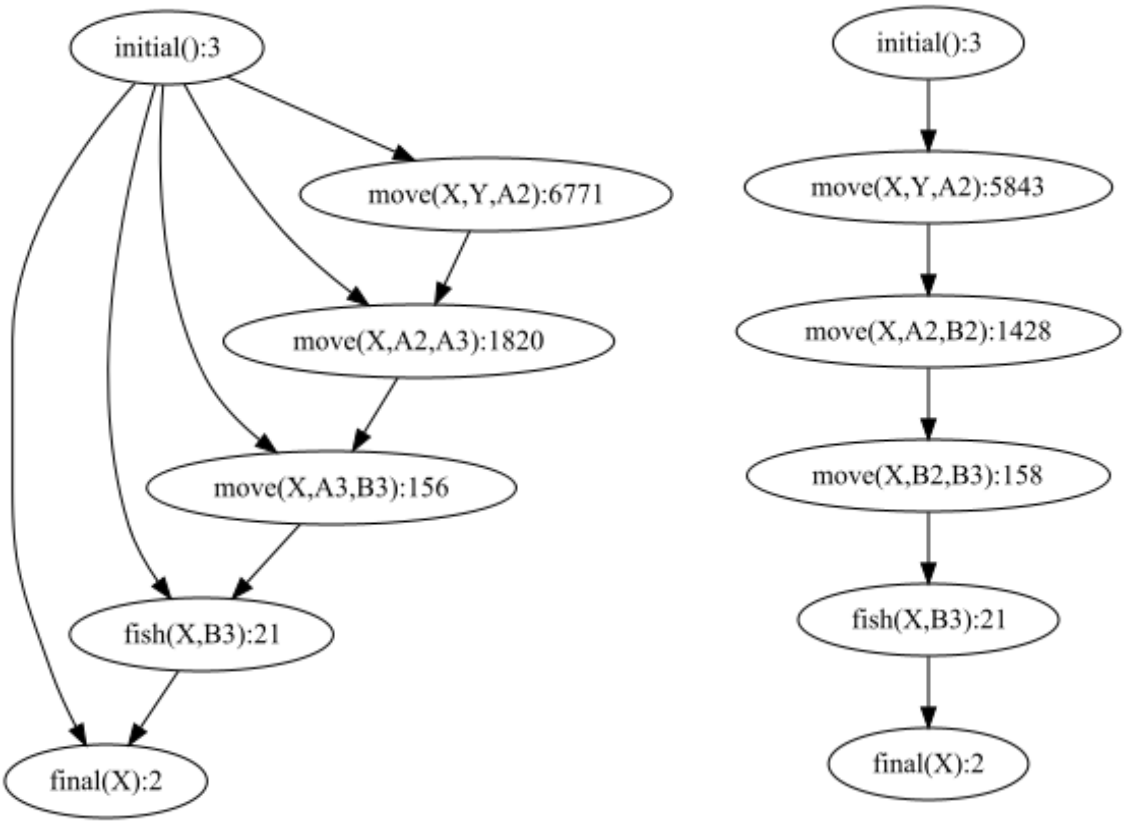
```
[WARN ] aStar_planning.pop.PopPlanningProblem - ___APPLY___ TO RESOLVE (OBLIGATION) :
 [member(X) , haveFish(X)] -> reportFish(X) IN Situation6
Add step: reportFish(X):495
Add situations: [Situation25, Situation26]
Add cc:
    [C25::X == C2::X, C38::X != C3::SELF, C495::X == C25::X]
Add tc:
    Situation25 < reportFish(X):495,
    reportFish(X):495 < Situation26,
    Situation6 < Situation25,
    Situation26 < Situation2
[INFO ] aStar_planning.pop.PopPlanningProblem - ___GOT___ PLAN #16
 - SITUATIONS : [Situation1, Situation2, Situation5, Situation6, Situation25, Situation26]
 - STEPS : [initial():3, final(X):2, fish(X,Z):25, reportFish(X):495]
 - CODENOTATIONS :
    C25::X == C2::X,
    C38::X != C3::SELF,
    C25::X == C2::X,
    C38::X != C3::SELF,
    C495::X == C25::X
 - TEMPORAL CONSTRAINTS :
    initial():3 < Situation1,
    Situation1 < Situation2,
    Situation2 < final(X):2,
    Situation5 < fish(X,Z):25,
    fish(X,Z):25 < Situation6,
    Situation6 < Situation2,
    Situation1 < Situation5,
    Situation25 < reportFish(X):495,
    reportFish(X):495 < Situation26,
    Situation6 < Situation25,
    Situation26 < Situation2
 - FLAWS :
    (OC): containsFishes(Z) IN Situation5
-- ORGANIZATIONS :
    [globalOrg, villageOrg, householdOrg]
[INFO ] aStar_planning.pop.PopPlanningProblem - HEURISTIC : 1
[INFO ] aStar_planning.pop.PopPlanningProblem - IS VALID : true
```

Figure 43: Résolution de l'action obligatoire de déclarer ses prises en ajoutant l'action obligatoire (en rouge), confirmée par l'absence de la lacune normative dans l'état suivant (en vert)

3.3.2. Comportements sous interdictions

3.3.2.1. Actions interdites

Pour le cas des actions interdites, si nous interdisons à l'agent de pêcher dans les zones protégées alors qu'il a pour obligation de trouver du poisson, nous obtenons le comportement décrit par la Figure 44(b) qui est comparé au plan normal sans interdiction de pêche sur tout objet ayant le rôle de "protégée" à moins d'avoir une licence.



(a) Plan à ordre partiel brut (b) Plan à ordre partiel simplifié

Figure 44: Comparaison des comportements obtenus avec et sans interdiction sur la pêche

Nous pouvons déduire les impacts suivants :

1) Plutôt que d'aller pêcher dans la zone A3 protégée, l'agent a décidé de pêcher dans la zone B3 à côté;
2) Il peut également considérer le fait d'aller chercher une licence pour continuer à pêcher dans A3, mais cette option est plus coûteuse en termes de planification car il

faut que l'agent se rende d'abord à un bureau de commune avant d'aller pêcher de nouveau. Celui-ci reste une deuxième option possible, mais plus coûteuse.

Pour comprendre ces deux impacts plus en détails, des fragments du parcours de l'espace de recherche sont présentés. Premièrement, la Figure 45 montre les traces d'exécution du planificateur qui détecte bel et bien une lacune normative : celle de pêcher dans une zone qui pourrait s'avérer une zone protégée.

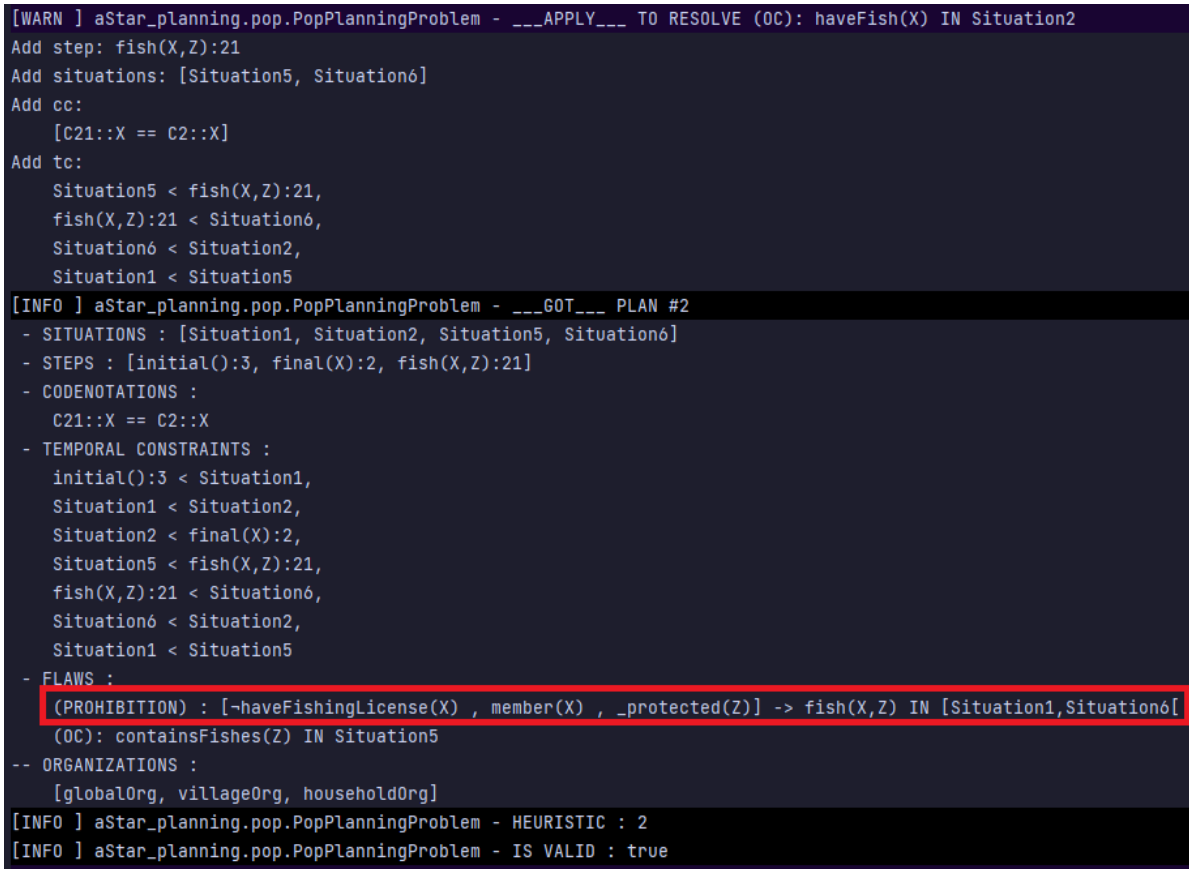


Figure 45: Trace d'exécution affichant une lacune normative de type interdiction encadrée en rouge

Son application dans l'intervalle de la *Situation1* (la situation initiale) et la *Situation6* (situation après l'action de pêcher) s'explique par le fait que :

- Dans *Situation1* la norme est applicable: l'agent n'a pas de licence de pêche, il est membre du village, et il existe bel et bien l'action interdite de pêcher après elle;
- Dans *Situation6* la norme reste également applicable, cependant, elle n'est pas violée, car il n'y a plus l'action interdite (pêcher) après la *Situation6*. Puisque l'interdiction est respectée, la lacune normative s'arrête à partir de la *Situation6*.

En faisant un *getOptions*, le planificateur propose différents opérateurs pour la résoudre, notamment sur la Figure 46 et la Figure 47, où les opérateurs possibles de nos contributions sont :

- La promotion;
- La démotion;
- Le contournement.

```
[INFO ] aStar_planning.pop.PopPlanningProblem - ______OPTIONS ARE______
[WARN ] aStar_planning.pop.PopPlanningProblem - --> OPTION #a TO RESOLVE (OC): containsFishes(Z) IN Situation5
Add cc:
    [C21::Z == C3::A3]
[WARN ] aStar_planning.pop.PopPlanningProblem - --> OPTION #b TO RESOLVE (OC): containsFishes(Z) IN Situation5
Add cc:
    [C21::Z == C3::B3]
[WARN ] aStar_planning.pop.PopPlanningProblem - --> OPTION #c TO RESOLVE (OC): containsFishes(Z) IN Situation5
Add cc:
    [C21::Z == C3::C3]
```

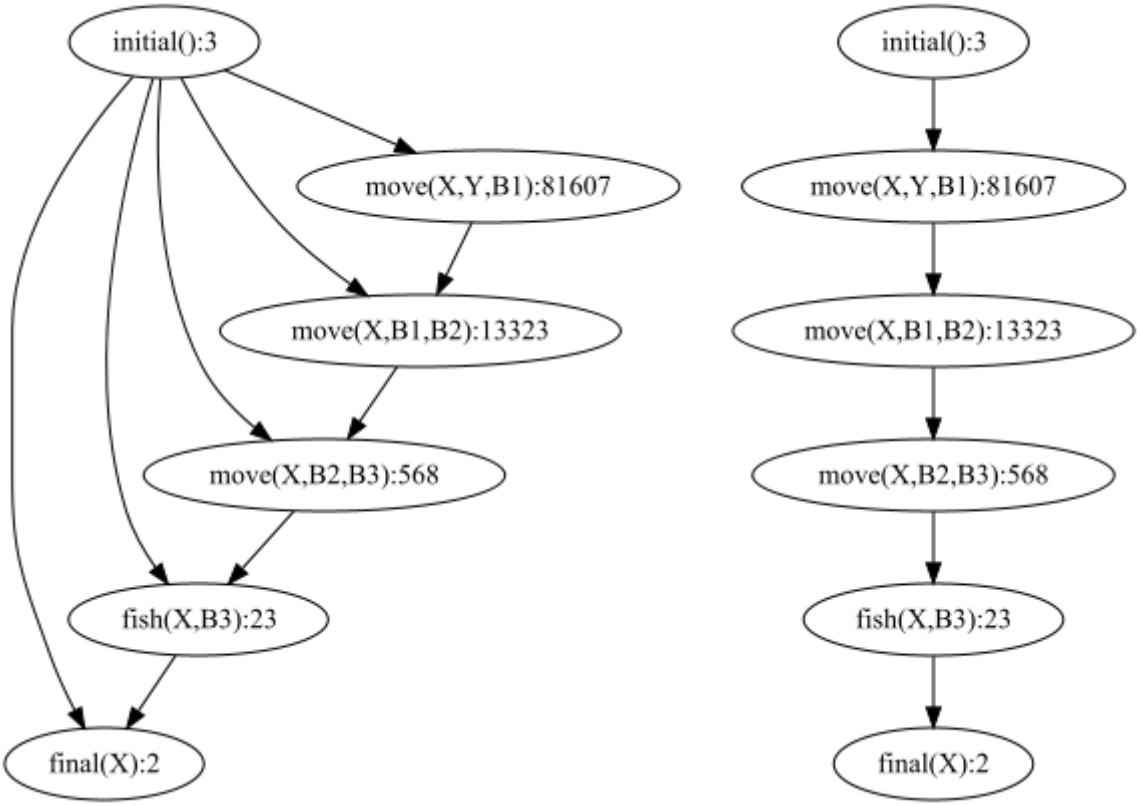
Figure 46: Les opérateurs classiques pour résoudre une condition ouverte classique

```
[WARN ] aStar_planning.pop.PopPlanningProblem - --> OPTION #d TO RESOLVE (PROHIBITION) : [¬haveFishingLicense(X)
 , member(X) , _protected(Z)] -> fish(X,Z) IN [Situation1,Situation6[
Add tc:
    Situation6 < Situation5
[WARN ] aStar_planning.pop.PopPlanningProblem - --> OPTION #e TO RESOLVE (OC): haveFishingLicense(X) IN Situation
Add step: getFishingLicense(X,Z):42
Add situations: [Situation7, Situation8]
Add cc:
    [C42::X == C21::X]
Add tc:
    Situation7 < getFishingLicense(X,Z):42,
    getFishingLicense(X,Z):42 < Situation8,
    Situation8 < Situation1,
    Situation1 < Situation7
[WARN ] aStar_planning.pop.PopPlanningProblem - --> OPTION #f TO RESOLVE (OC): ¬member(X) IN Situation1
Add cc:
    [C21::X != C3::SELF]
[WARN ] aStar_planning.pop.PopPlanningProblem - --> OPTION #g TO RESOLVE (OC): ¬_protected(Z) IN Situation1
Add cc:
    [C21::Z != C3::A3]
[WARN ] aStar_planning.pop.PopPlanningProblem - --> OPTION #h TO RESOLVE (OC): ¬_protected(Z) IN Situation1
Add cc:
    [C21::Z != C3::B1]
```

Figure 47: Les opérateurs proposés par le planificateur pour résoudre une lacune normative sous la forme d'une proposition interdite manquante.

3.3.2.2. Atomes interdits

Pour le cas des atomes interdits : si nous interdisons à l'agent d'être dans une zone sacrée, nous obtenons le comportement de la Figure 48.



a) Plan à ordre partiel brut b) Plan à ordre partiel simplifié

Figure 48: Comportement obtenu en interdisant le passage dans une zone sacrée.

Le chemin le plus évident pour trouver du poisson (bouger de A1, A2 et A3) illustré dans la Figure 39 (a) présente désormais une lacune en plus, ce qui laisse la priorité aux autres états alternatifs. Notamment celui d'aller pêcher dans d'autres zones, ou, encore sur A3, mais en n'incluant pas un chemin passant par des zones sacrées. La présence de cette lacune est démontrée sur la Figure 49.

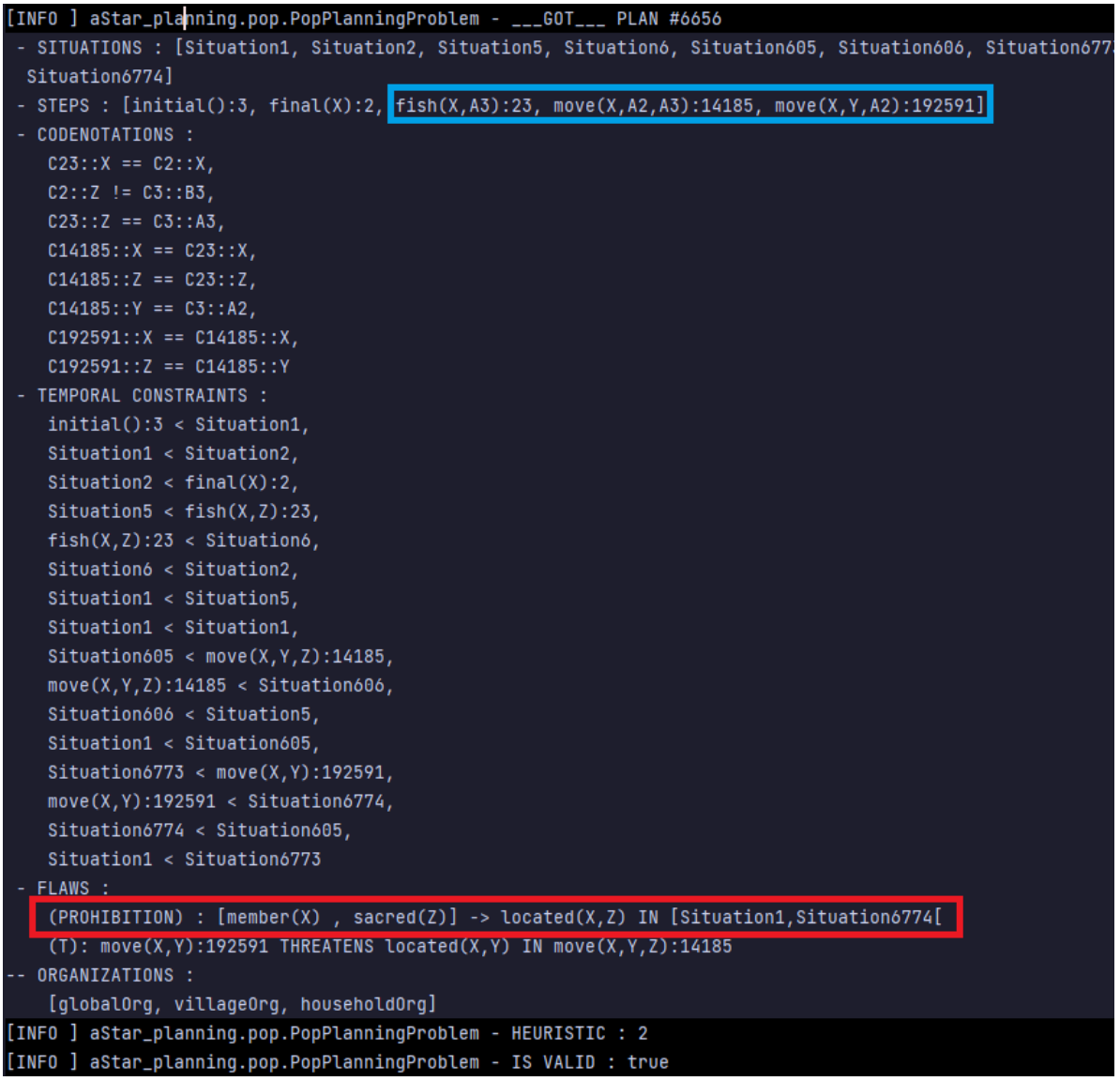


Figure 49: Apparition d'une lacune normative (en rouge) pour le chemin classique A1-A2-A3 au vu du passage en zone sacrée A2 (en bleu)

Nous noterons qu'il est également impossible de contourner cette norme, puisqu'il n'existe pas d'action pour quitter le fait d'être membre du village, ou pour désacraliser une zone. Il est vrai que le planificateur peut toujours mettre une contrainte de non-codénotation entre X et l'agent lui-même, mais cela reviendrait à dire que le plan qu'il conçoit ne sera plus le sien, et qu'il doit trouver quelqu'un d'autre pour exécuter ce plan. La notion de coopération, ou de mission à confier à d'autres agents est toutefois hors de la portée de ce travail.

3.3.3. Comportements sous permissions

3.3.3.1. Action permise

En ajoutant une permission qui incombe à tout membre du village qui décrit que la coupe de bois est permise seulement quand on a une licence d'exploitation qui est délivrée par le bureau communal, le planificateur en déduit le plan sur la Figure 50: l'agent va d'abord chercher une licence avant d'aller couper du bois.

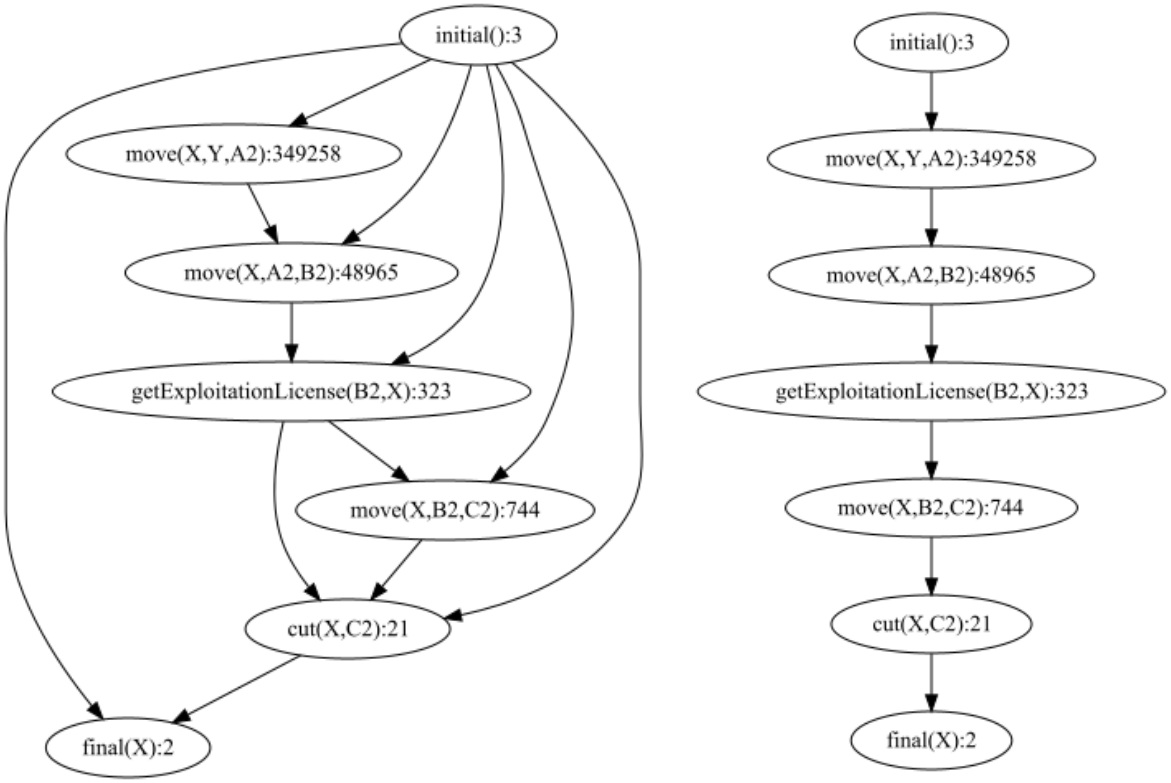


Figure 50: Plan obtenu quand la coupe est permise seulement avec une licence d'exploitation

Dans la Figure 51, nous pouvons voir qu'après avoir ajouté l'action de couper (*cut*), une nouvelle lacune apparaît : la condition ouverte d'avoir une license d'exploitation (*getExploitationLicense*) qui est devenue une des préconditions de l'action *cut* dans la situation qui précède la coupe.

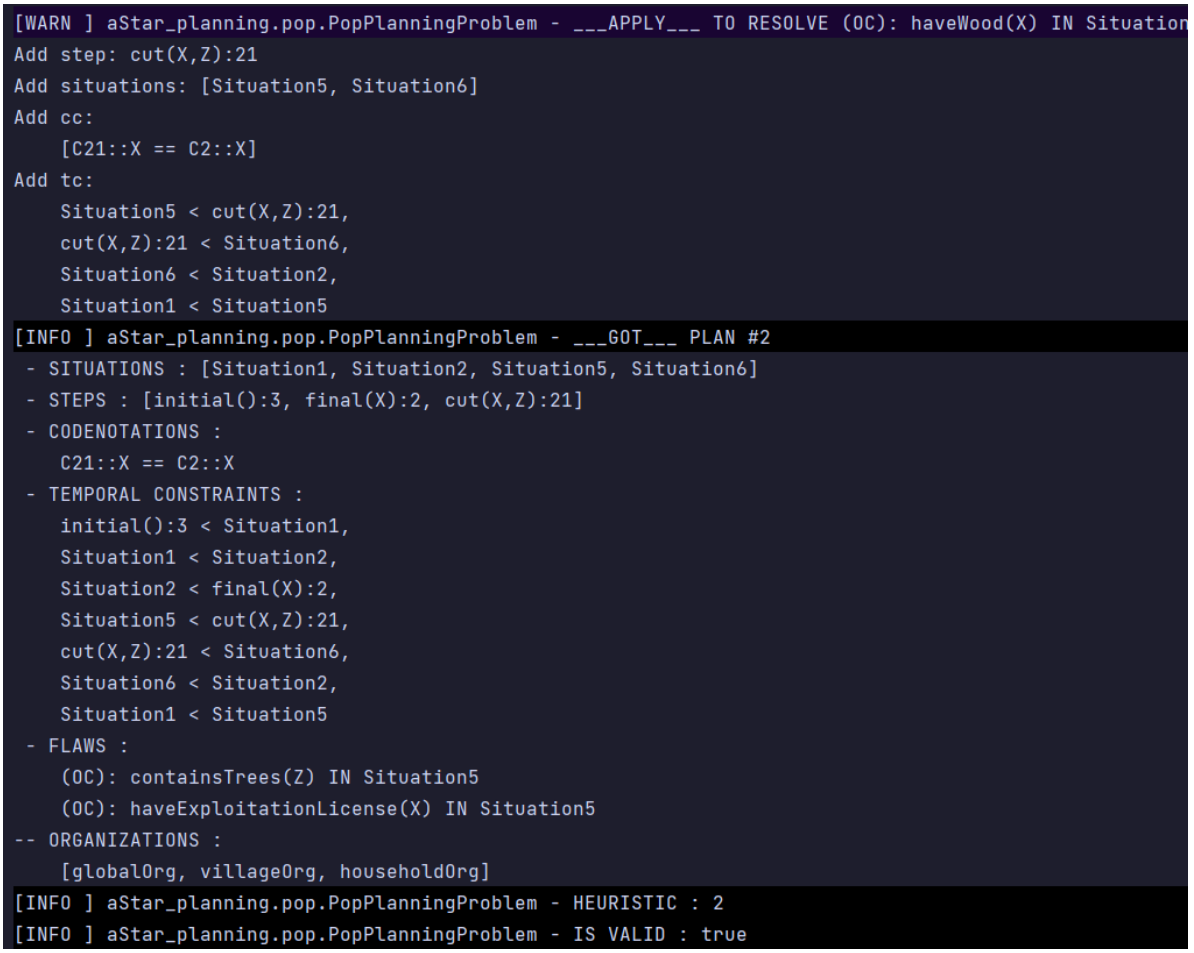


Figure 51: Trace d'exécution du planificateur où une norme stipulant qu'une coupe est permise quand l'agent a une licence d'exploitation, ce qui crée la condition ouverte haveEpxploitaitonLicense.

Celle-ci requiert alors à son tour de se rendre au bureau communal, situé dans la zone B2 au centre de la carte. Pour pouvoir obtenir une licence d'exploitation avec le prédicat *getExploitationLicence(X, Z),* il faut en effet deux préconditions :

- *located(X, Z) :* être dans une certaine zone Z;
- *office(Z)* : la zone Z en question est un bureau communal.

Pour la première précondition, l'agent se trouve déjà dans une zone, nous pouvons donc dire qu'il va acquérir sa licence d'exploitation dans cette zone avec une contrainte de codénotation *geExploitationLicence(X, Z = A1)*, mais A1 n'est pas un bureau communal, et il reste donc la condition ouverte de rendre nécessairement vraie que A1 est un bureau communal, comme le démontre la lacune de la Figure 52.

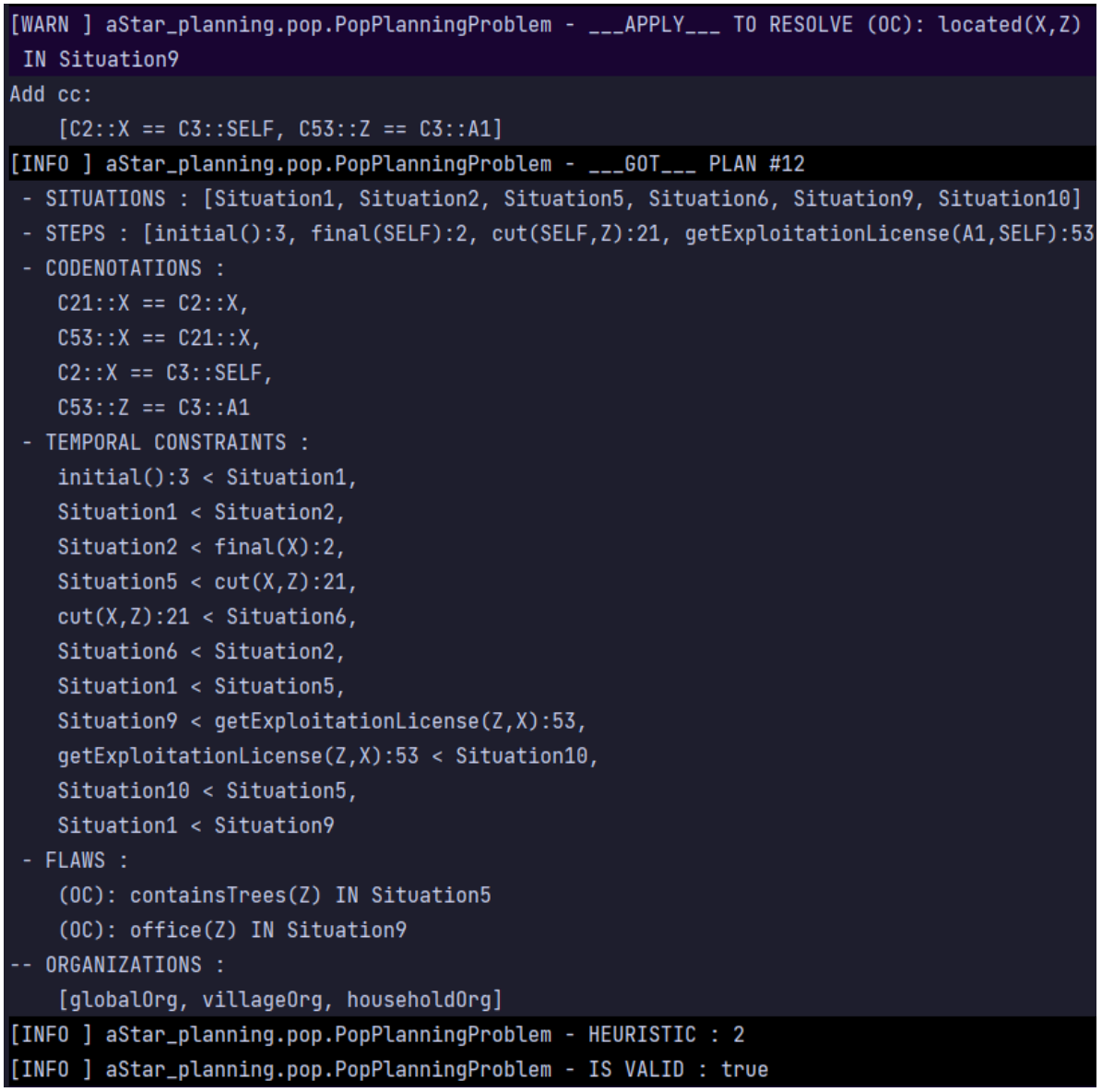


Figure 52: Progression de l'espace de recherche pour trouver un chemin permettant d'obtenir une licence d'exploitation pour pouvoir couper.

Dans notre modèle, nous n'avons pas d'actions qui permettent de modifier la topographie de l'environnement, donc cette lacune est insolvable, à moins de changer de zone, laissant donc le chemin pour d'autres parcours de l'espace de recherche.

3.3.3.2. Atome permis

Pour le cas des atomes permis, c'est-à-dire, les permissions qui rendent permis certains atomes, elles seront traduites en une interdiction. Pour cette implémentation, la norme de l'institution villageoise décrivant que "si l'agent a un licence de pêche, alors il est permis d'avoir un filet de pêche" sera traduite en une interdiction "si l'agent n'a pas de licence de pêche, alors il est interdit d'avoir un filet de pêche". Avec la syntaxe du langage nous aurons donc initialement la permission :

```
if (haveFishingLicense(agent)) then permission(hasFishingNet(agent))
```

Conformément à nos contributions, cette permission génère l'interdiction suivante :

```
if (not haveFishingLicense(agent)) then prohibition(hasFishingNet(agent))
```

Pour rendre le fait d'avoir un filet de pêche pertinent, il sera ajouté pour ce scénario aux préconditions de l'action *fish(agent, zone)* qui sera ainsi décrite par le Code 18. L'agent doit donc avoir un filet pour pêcher, ce qui est permis quand il a une licence de pêche.

```
Action fish(var X, var Z){
      Preconditions{
            global.located(X,Z), haveFishingNet(X), hasFish(Z),
            hasFishingNet(X)
      },
      Postconditions {
            not village.hasFish(Z),haveFish(X)
      }
}
```

Code 18 : Modification de l'action de pêche pour y inclure la nécessité d'un filet de pêche.

Pour rendre compte de cette conversion, nous proposons de retracer l'ensemble des normes applicables à l'agent avec une trace d'exécution des permissions actuelles, et des interdictions actuelles au sein du plan initial dans la Figure 53.

```
"C:\Program Files\Java\jdk-17.0.2\bin\java.exe" ...
[DEBUG] aStar_planning.pop_with_norms.components.NormativePlan - -> Prohibitions before: []
[DEBUG] aStar_planning.pop_with_norms.components.NormativePlan - -> Permissions are:
 [(PERMISSION) : [haveFishingLicense(X)] -> hasFishingNet(X)]
[DEBUG] aStar_planning.pop_with_norms.components.NormativePlan - -> All prohibitions after:
 [(PROHIBITION) : [¬haveFishingLicense(X)] -> hasFishingNet(X)]
[INFO ] aStar_planning.pop_with_norms.NormativePlanningProblem - INITIAL STATE : PLAN #1
 - SITUATIONS : [Situation1, Situation2]
 - STEPS : [initial():3, final(X):2]
 - CODENOTATIONS :
    NONE
 - TEMPORAL CONSTRAINTS :
    initial():3 < Situation1,
    Situation1 < Situation2,
    Situation2 < final(X):2
 - FLAWS :
    (OC): haveFish(X) IN Situation2
-- ORGANIZATIONS :
    [globalOrg, villageOrg, householdOrg]
```

Figure 53: Trace d'exécution des méthodes pour convertir les permissions en interdictions

Une fois cette interdiction générée, elle est traitée au même titre que les atomes interdits classiques, et donne lieu au plan à ordre partiel sur Figure 54 et Figure 55.

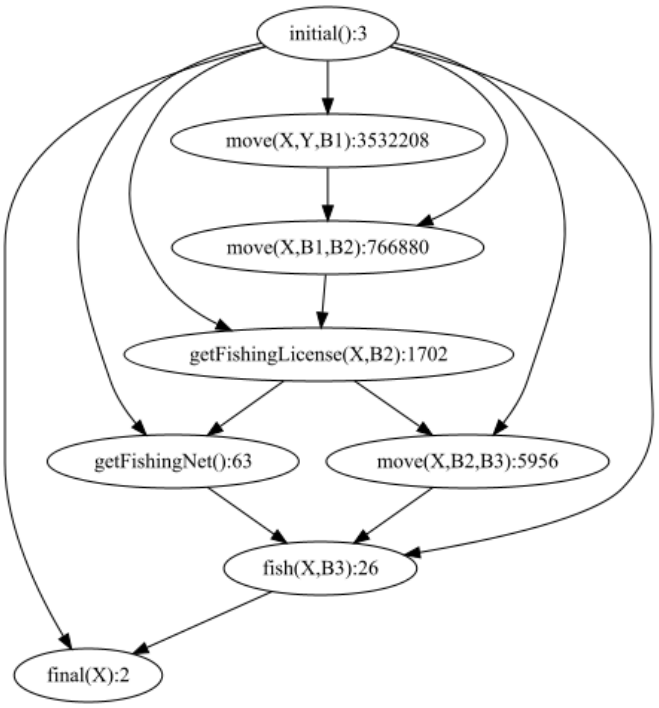


Figure 54: Plan obtenu où si l'agent a une licence, alors il est permis d'avoir un filet de pêche.

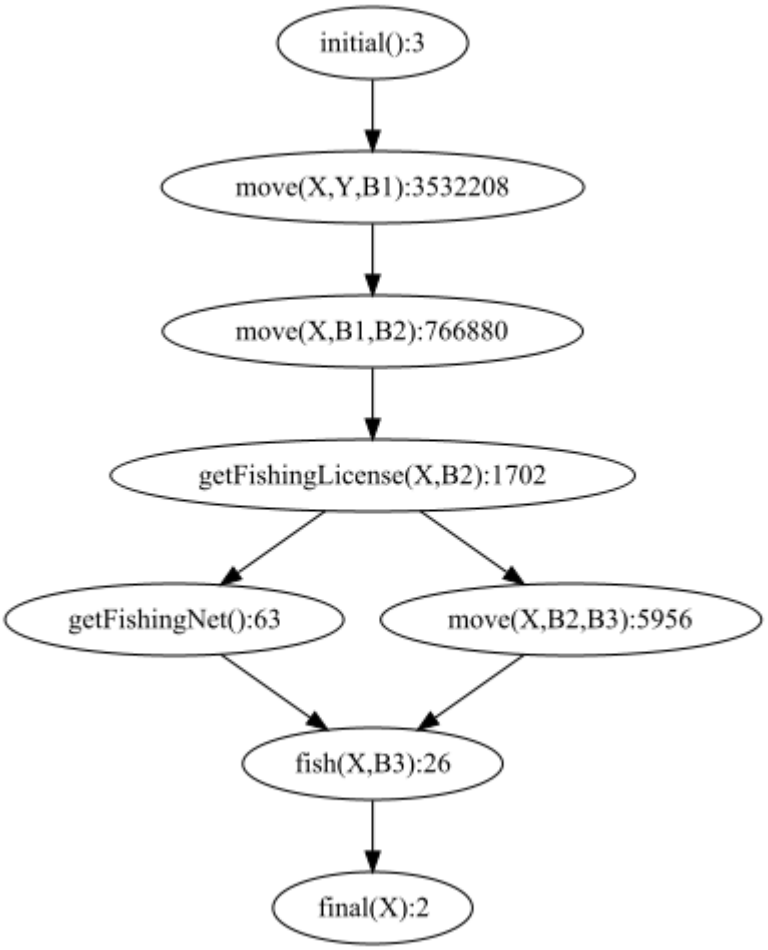


Figure 55: Plan simplifié obtenu avec la permission : si on a une licence de pêche, alors il est permis d'avoir un filet de pêche.

En effet, par une stratégie de contournement, l'agent prend en compte cette interdiction en rendant nécessairement vraie la négation de la condition $\neg haveFishingLicense(X)$ qui devient alors une condition ouverte classique dont la résolution est représentée par la Figure 56. Cette condition ouverte sera ensuite traitée en ajoutant l'action qui permet d'obtenir le droit d'avoir un filet de pêche.

```
[WARN ] aStar_planning.pop.PopPlanningProblem - ___APPLY___ TO RESOLVE (OC):
 haveFishingLicense(X) IN Situation35
Add step: getFishingLicense(X,Z):3509104
Add situations: [Situation97949, Situation97950]
Add cc:
    [C3509104::X == C26::X]
Add tc:
    Situation97949 < getFishingLicense(X,Z):3509104,
    getFishingLicense(X,Z):3509104 < Situation97950,
    Situation97950 < Situation35,
    Situation1 < Situation97949
```

Figure 56: Trace d'exécution de l'opérateur qui permet de résoudre la lacune par contournement

Pour les autres opérateurs que sont 1) la promotion et 2) la démotion, elles sont considérées, mais ne donnent pas de plans cohérents, puisque l'intervalle d'applicabilité de la permission s'étend sur la totalité du plan : de la situation initiale à la situation finale, et aucun élément du plan ne peut se mettre avant la pas initial, tout comme aucun autre élément du plan ne peut se mettre après le pas final.

4. Discussions

La question centrale de cette partie est de discuter de ce qu'apportent nos résultats, en comparaison à ce qui a été cité dans l'état de l'art, ainsi que les limites de nos contributions, afin de pouvoir dresser les sujets de travaux futurs. Pour cela, nous reprenons quelques travaux illustrés dans le Tableau 8 synthétisant l'état de l'art en y intégrant notre approche.

4.1. Par rapport aux types d'agents

Pour des approches telles que BOID [Broersen et al., 2021], lorsque des objectifs et des normes sont en conflit, les agents font des décisions statiques en fonction de leur type (égoïste, social, rebelle…). Toutefois, les stratégies de l'agent pour prendre des décisions restent statiques car ils ne peuvent changer de type au cours de la simulation, et les agents adoptent les mêmes préférences en toute situation: l'explicabilité des décisions se résume donc à son type. La spécification de "comment" les agents accomplissent les objectifs d'une norme est par ailleurs très implicite, puisque les auteurs se focalisent sur quel objectif choisir en tant qu'intention de l'agent.

Dans le cas de notre contribution, la résolution de ces conflits se traduit par la notion d'une lacune dans le plan de l'agent, et se résout avec plusieurs stratégies explicites, et qui sont toutes considérées tant qu'elles sont valides, fournissant des informations explicites sur les options, et pourquoi telle option est meilleure qu'une autre. De plus, puisque la violation est envisagée seulement quand aucun plan qui respecte l'ensemble des normes n'est trouvé, nos contributions permettent de prioriser plus facilement des plans qui prennent en compte les normes et qui accomplissent les objectifs de l'agent.

Tableau 8 : Synthèse de nos contributions en comparaison à la littérature.

Travail	Modalité déontique	Objectifs des normes	Stratégies de prise en compte des normes	Planification automatisée	Sanction et récompenses	Raisonnement quantitatif
Ramarozaka et al., 2021 Ramarozaka et al., 2022	Obligation, Interdiction, Permission	État Action	Planification: détection et correction des lacunes normatives du plan. Contournement : rendre une norme non-applicable	OUI (POP étendu)	NON	NON
Broersen et al., 2001 (BOID)	Obligation	État	Typage fixe : type des agents	NON	NON	NON
Panagiotidi et al., 2013	Obligation, Interdiction	État	Réparation : chaque violation introduit la nécessité de le réparer	OUI	NON	OUI
Kammler et al., 2022	Obligation, Interdiction, Permission	Action État	Planification avec la notion de valeurs et des points de contrôle	OUI (GOAP)	OUI	OUI (Sur les jauges)
Lopez & Marquez, 2004	Obligation, Interdiction	État	Typage fixe : type des agents avec la notion d'importance et de motivation	NON	OUI	OUI
Meneguzzi & Luck, 2009 (Normative AgentSpeak(L))	Obligation, Interdiction	Action État	Modification de la bibliothèque de plans Planification et libraire de plans	OUI (STRIPS)	NON	NON
Viana et al., 2015 (JSAN)	Obligation, Interdiction, Permission	Action État	Typage fixe : type d'agent extensible par des classes Java	NON (Plans prédéfinis)	OUI	OUI (récompenses et les sanctions)

4.2. Par rapport aux raisonnements quantitatifs

Les travaux de [Lopez & Marquez, 2004] ajoutent des améliorations aux types d'agents en considérant les apports des sanctions et des récompenses des normes et en les comparant avec d'autres objectifs. Notamment les notions d'importance et de motivation sont ajoutées. Un des désavantages de ce type d'approche reste le comportement prédéfini des agents qui est statique tout au long de la simulation, malgré l'ajout du critère de motivation: l'action ou la norme définie dont l'importance répond au type prédéfini de l'agent (opportuniste, craintif, pressurisé …) sera toujours choisie.

Pour remédier aux décisions statiques, des approches comme ANA [dos Santos Neto et al, 2012] dynamisent les motivations de l'agent au cours de la simulation. D'autres approches n'utilisent pas de type d'agents mais exclusivement des fonctions d'utilité, comme dans EMIL-A [Andrighetto et al, 2012], avec la saillance des normes. Cependant, il faut toujours fournir des informations spécifiques au sujet d'étude : l'utilité des actions, des récompenses, des sanctions sous une forme quantifiable qui doit être prédéfinie par le modélisateur, alors que définir une fonction d'utilité pour toutes les éventualités possibles n'est pas faisable.

Les contributions du présent travail ne requièrent pas de quantifier les fonctions d'utilité de tous les scénarios possibles, ni de leur motivation ou de leur importance, mais nos contributions adoptent plutôt une fonction d'utilité estimée automatiquement à travers la fonction heuristique générique, qui met l'accent soit sur la longueur de la recherche, ou le nombre de lacunes dans le plan.

4.3. Par rapport aux plans prédéfinis

La plupart des approches de l'architecture BDI: telles que dMars [d'Inverno et al., 1998] ou les programmes AgentSpeak [Bordini, Hübner, 2005], se reposent sur une bibliothèque de plans prédéfinis, ce qui empêche l'agent de trouver de nouveaux plans face aux normes, y compris les approches normatives tel que JSAN [Viana et al., 2015] ou NoA [Kollingbaum & Norman, 2003].

Ce procédé crée une dépendance entre les comportements produits et l'exhaustivité du modélisateur dans sa spécification des plans possibles, d'autant plus que dans un monde ouvert, de nouvelles normes peuvent être appliquées.

Dans nos contributions, les agents ont la possibilité de générer des plans d'actions de zéro, permettant ainsi la découverte de nouveaux comportements possibles. De plus, pour prendre en compte les normes qui apparaissent au cours de la simulation, la notion d'organisation permet à l'agent d'acquérir une ontologie, y compris des actions, pour qu'il puisse accomplir le rôle qu'il joue dans cette organisation. Des plans prédéfinis peuvent par ailleurs être utilisés en parallèle à la planification proposée : si un plan prédéfini marche, l'exécuter, sinon, invoquer le planificateur et ajouter le plan trouvé dans la bibliothèque de plans, à l'instar des travaux de [Meneguzzi & Luck, 2009], mais qui n'implémente pas toutefois la notion d'organisation, et prend en compte uniquement les interdictions au niveau de la bibliothèque de plans, et qui ne prend pas en compte non plus les permissions.

4.4. Par rapport aux autres planificateurs

Les travaux de [Kammler et al., 2022] réalisés dans la même période que le présent travail présentent des apports similaires en se basant sur une version plus riche de la grammaire utilisée dans le présent travail à travers ADICD1RO, un système de valeurs, et un planificateur basé sur des points de contrôle en chaînage arrière, au lieu d'utiliser des actions atomiques décrites dans nos contributions.

Un des avantages du planificateur de [Kammler et al., 2022] est d'avoir un typage d'agent dynamique comme proposé dans ANA [dos Netos Santos et al., 2012] tout en ayant la possibilité d'utiliser un planificateur à la STRIPS [Fikes & Nilsson, 1971], et un système de préférence plus flexible, entre dix valeurs.

La délibération moyenne permet également de trouver des plans d'actions alternatives lorsque l'action planifiée n'est plus exécutable, tandis que la délibération complexe fait appel au planificateur GOAP [Orkin, 2006] permet de choisir l'objectif dont la satisfaction est la plus grande, et crée un plan pour atteindre cet objectif.

Enfin, puisque le planificateur se base sur le concept de jauge pour vérifier quelles sont les normes les valeurs les plus importantes, et in fine, quelles sont les normes les plus urgentes, le planificateur utilise des quantités pour les prioriser. [Kammler et al, 2022] offre donc une plus grande expressivité, et les agents peuvent prendre des décisions en alignement avec leurs valeurs: il faut donc associer des valeurs à chaque norme.

Le présent travail n'inclut pas la notion de valeur, ni de quantité, ni de priorisation des normes. Comparé au planificateur utilisé par [Kammler et al., 2022], nos contributions offrent une liste plus explicite des stratégies pour prendre en compte une norme. Plus particulièrement une stratégie de contournement est spécifiée, qui est quasiment absente dans les autres approches à notre connaissance actuellement.

Cette liste explicite permet une plus grande explicabilité, tout en maintenant le nombre d'entrées à fournir plus accessible : pour nos contributions, il faut décrire les organisations, les institutions, leurs actions, les normes, tandis que pour [Kammler et al., 2022] il faut théoriquement se familiariser avec la notion de valeur, un minimum pour classifier les valeurs de manière adéquate, et expliquer le comportement obtenu à des thématiciens en se basant sur la notion de jauge.

4.5. Par rapport aux concepts utilisés

Dans le choix d'une représentation des normes, nous pouvons constater que la plupart des architectures d'agent normatives et des langages dits "orientés-agents" divergent également pour la plupart des concepts originaux de l'économie institutionnelle, mais reprennent des concepts similaires. Par exemple, [Lopez & Marquez, 2004; dos Santos Neto et al., 2012] nous pouvons voir les similarités entre : Adressé et Attribut, contexte et les conditions, récompenses, ou encore dans [Panagiotidi et al., 2013] entre les concepts de type et de modalité déontique.

Nos contributions se basent plutôt sur les notions d'institution exprimées à travers la grammaire institutionnelle dans ADICO [Crawford & Ostrom, 1995], ce qui permet une accessibilité et une inclusion des thématiciens, et potentiellement de migrer le vers d'autres grammaires institutionnelles plus riches basé sur ADICO.

Toutefois, d'autres approches intègrent également récemment les institutions dans les SMA telles que les institutions et les organisations dans [Tomic et al., 2018; Fornara et al., 2013], ou les notions des normes constitutives dans [Alechina et al., 2018].

Au vu des notions techniques utilisées, il faut également offrir un moyen aux thématiciens pour construire les indicateurs souhaités sur l'efficacité des normes (revenus des foyers, taux de coupe, etc.). Des langages dédiés convergent vers cet objectif notamment avec les travaux de [Rakotonirainy et al., 2015b] qui peuvent être appliqués à nos contributions pour leur offrir une stratégie d'observation adaptée aux discours des thématiciens.

4.6. Limites

Parmi les limites du présent travail [Ramarozaka et al., 2021], nous pouvons en dénombrer quatres (04) qui seront présentées dans cette section:

1) La non prise en compte des quantités dans la planification;
2) La non prise en compte du temps et de l'espace;
3) Le problème de la grande complexité du planificateur hérité du POP;
4) La non prise en compte de l'evolution des normes et des rôles.

4.6.1. Prise en compte des quantités

Un des aspects importants dans toute prise de décision est la notion de quantité. Par exemple, un agent qui veut amasser 100 poissons ne va pas forcément exhiber le même comportement qu'un autre agent qui veut en amasser un seul : il va privilégier les actions qui lui rapportent le plus de poissons; de la même manière, un agent qui risque une amende / récompense de 10.000.000 Ariary[17] ne va pas forcément engendrer le même comportement s'il risquait une amende / récompense de 1.000 Ariary.

[17] L'Ariary est la monnaie locale à Madagascar.

Nous ne prenons pas en compte ces quantités dans les exemples de problèmes de planification présentés, mais elles peuvent être représentées sous la forme de prédicats du premier ordre.

4.6.2. La non prise en compte de l'espace et du temps

Dans nos contributions, nous nous basons sur le POP où nous avons un ensemble de situations représentant une description discrète (ou instantanée) du monde. De ce fait, il manque encore au planificateur de délimiter la notion d'intervalle de temps dans le plan à ordre partiel. Par exemple : comment peut-on représenter des affirmations telles que "avant de pêcher", "durant l'été", "à 8h du matin", dans ce type de planificateur. La notion d'intervalle dans la prise en compte des interdictions pour obtenir l'intervalle des situations où la norme est applicable constitue une première approche pour lier intervalles de temps et POP, mais demeure insuffisante.

De même pour l'espace, que nous n'intégrons pas encore les coordonnées spatiales de l'agent dans les plans. Le planificateur ne peut pas encore décrire dans telle situation où se trouve l'agent dans l'espace, si ce n'est avec des prédicats *located(X, Y)*. L'agent ne peut de ce fait pas interagir avec des cartes thématiques telles que des images satellitaires qui peuvent leur faire office d'environnement. Ces deux aspects ont déjà été abordés dans [Raharivelo & Müller, 2018] dans le calcul de l'applicabilité des normes, et peuvent être rattachés au présent travail pour de futurs travaux.

4.6.3. La complexité des extensions du POP

Le POP original de [Chapman, 1987] avec le planificateur TWEAK est polynomial, toute extension y est décrit comme étant NP-complet y compris le planificateur proposé. En effet, l'espace de recherche grandit encore plus au vu des nouvelles lacunes possibles. L'heuristique générique ainsi que l'algorithme basique A* ne sont pas optimisés pour un usage pratique impliquant plusieurs agents: dans le cadre des tests du modèle IRIELA sur un

ordinateur portable core i9 cadencé à 2.9 Ghz et une mémoire vive de 16 Go , le planificateur prend entre 3 minutes et 50 minutes pour trouver une solution qui accomplit 1 ou 2 normes.

Les plus longs tests se caractérisent par les scénarios où l'agent veut du poisson, mais qu'il doit se rendre autre part avant pour obtenir une licence, comme la norme l'oblige : les chemins pour passer dans les deux zones même dans l'espace d'une grille en 3x3 produit un facteur de branchement non négligeable de l'espace de recherche.

4.6.4. La non-évolution des normes et des rôles

Dans le cadre de nos simulations, nous prenons comme hypothèse que les normes restent fixes, c'est-à-dire que la modification autonome des normes de l'agent au cours de la simulation ne sont pas prises en compte dans le présent travail, bien que celle-ci constitue un atout majeur pour les thématiciens. De la même manière, dans nos simulations, nous n'avons pas d'actions qui permettent à un agent de quitter le rôle qui lui a été assigné, ou de le déléguer à quelqu'un d'autre.

5. Perspectives

5.1. Perspectives en ingénierie

5.1.1. Création d'autres modèles tests

Un des travaux qui pourrait être entrepris serait de tester l'outil de manière concrète en parallèle à une modélisation à base d'agent standard, comme ce fut le cas dans la commune de Didy et la communauté locale d'Antontona avec les modèles MIRANA [Aubert et al., 2010] et HINA [Aubert & Müller, 2013].

À travers le problème de planification IRIELA, nous pouvons dresser le type de problématique que l'outil permet d'aborder. Nous pouvons par exemple suggérer l'utilisation de l'outil dans les thématiques suivantes :

- La gestion durable des ressources forestières;
- La régulation de l'empreinte carbone par rapport aux activités humaines;
- La gestion durable de ressources naturelles marines renouvelables;

5.1.2. Intégration dans des plateformes de simulation

Comme toute application informatique, la mise en place d'interfaces graphiques plus simples à utiliser est également de mise, pour obtenir une plateforme de modélisation complète à disposition des juristes, des agronomes et des modélisateurs.

Etant donné qu'il existe déjà des plateformes de simulation à base d'agents, nous pouvons également prévoir la mise en place d'un plug-in pour intégrer le langage vers des plateformes ou framework agents comme CORMAS [Bousquet et al., 1998], GAMA [Taillandier et al., 2012], JASON [Bordini & Hübner, 2006] ou NetLogo [Tisue & Wilensky, 2004]. On constate cette interopérabilité d'outils de modélisation surtout à travers JaCaMo de [Boissier et al., 2016].

5.1.3. Amélioration de la performance

Un des domaines que l'on peut améliorer est celui de la performance, qui est le problème commun aux grands espaces de recherche. Même dans ses versions les plus basiques, les extensions sur les problèmes de planification sont connues pour être NP-completes [Erol et al., 1992]. Plusieurs pistes pour simplifier la complexité de la planification ont été exploitées et peuvent toujours être rajoutées en conjonction au planificateur proposé dans ce travail.

Une des premières améliorations serait de distinguer les différentes lacunes et les différents opérateurs dans le calcul heuristique : c'est-à-dire, rendre plus "importante" une type de lacune spécifique, et rendre plus coûteuse une modification par rapport à une autre, et comparer la performance de ces différents paramètres.

Une autre option est de remplacer l'algorithme A*, en effet, parce que l'outil est découplé de l'algorithme de recherche, il peut être remplacé aisément par un algorithme plus efficace. Une approche que nous pouvons considérer est l'utilisation d'un algorithme "*anytime*" [Dean & Boddy, 1998; Horvitz, 1987] qui peut être arrêté à tout moment, et fournir un plan, bien que sous-optimal. La qualité du plan peut être améliorée en laissant le planificateur tourner plus longtemps et en lui fournissant plus de ressources. Par exemple, nous avons l'algorithme anytime A* [Likhachev et al., 2005] qui est similaire à A* mais qui peut être arrêté à tout moment, en gardant traces des plans partiels les plus prometteurs.

Plus loin, malgré le fait que l'architecture calcule un plan de zéro, il n'est pas exclu que l'on y intègre un ensemble de plans pré-compilés, que le planificateur peut prendre comme état initial de la planification, et inversement, chaque plan obtenu pour atteindre un certain objectif peut être stocké dans un ensemble de plans et être exploité par d'éventuels problèmes de planification ou par des algorithmes d'apprentissage. Selon [Bacchus & Kabanza, 2000], un planificateur vraiment efficace doit recourir à plusieurs mécanismes.

5.1.4. Evaluation et évolution de l'outil

Une autre perspective en ingénierie serait de collecter un retour utilisateur pour évaluer l'ergonomie du langage, c'est-à-dire l'intuitivité de la syntaxe concrète, et de l'interface pour écrire avec le langage, ainsi que de la visualisation de l'espace de recherche pour comprendre le comportement généré: la mise en place d'une interface graphique, notamment pour visualiser les différentes lacunes au cours de la planification, facilite l'exploration des résultats.

Une autre manière d'explorer les résultats obtenus de ce planificateur est celle similaire à JaCaMo web [Amaral & Hübner, 2019] où il devient possible de changer les paramètres de la simulation au cours de l'exécution : ceci permet d'ajouter de nouvelles

normes à la volée, et de voir comment l'agent réagit en conséquence, prodiguant une meilleure expérience en termes d'exploration des résultats, et un outil de débogage en même temps.

5.2. Perspectives de recherche

5.2.1. Prise en compte des sanctions et récompenses

Le langage et le planificateur que nous avons mis en place tentent de trouver un plan pour satisfaire toutes les normes applicables à l'agent, sans tenir compte des sanctions : les récompenses et les punitions. Cependant, un aspect à considérer est que l'agent peut prioriser l'accomplissement d'une norme pour acquérir sa récompense qui converge vers ses objectifs personnels, ou à contrario, qui priorise l'accomplissement d'une norme pour éviter une punition qui diverge avec ses objectifs personnels.

L'approche globale est d'inclure les sanctions et les récompenses dans les normes régulatrices comme pour les travaux de [Kammler et al., 2022; Viana et al., 2015; Lopez & Marquez, 2004] sous une forme quantitative, pour dresser une préférence entre prendre en compte la norme et la violer.

À cela peut s'ajouter l'ajout d'un opérateur supplémentaire que l'agent peut considérer au cours du POP: violer la norme, et évaluer l'heuristique des nouveaux états obtenus. Il s'agit d'envisager une planification violant le moins de normes possibles à un stade plus précoce si celle-ci est plus optimale, plutôt que de considérer la violation des normes seulement quand il n'y a aucun plan qui permet de satisfaire toutes les normes.

5.2.2. Émergence et évolution des normes

Un des horizons explorables dans la prise en compte des normes est la notion d'innovation des normes, c'est-à-dire des normes qui ne sont plus limitées à une description statique mais qui peuvent changer au fil de la simulation : les comportements produits par les agents peuvent faire émerger de nouvelles normes, ou modifier les normes existantes, voire les abolir.

5.2.3. Prise en compte de l'espace, du temps et des quantités

Un des aspects importants dans "comment les agents vont respecter ou violer les normes" est le fait de dire "quand" et "où" il doit le faire. Il est nécessaire de pouvoir exprimer des normes spatio-temporelles, par exemple : "Pour tout agriculteur dans la commune urbaine de Fianarantsoa, il est interdit de couper du bois pendant la période de sécheresse".

Les travaux récents dans [Raharivelo & Muller, 2018] ont proposé un outil pour pouvoir les exprimer, et en déduire l'ensemble des normes applicables à un agent, dans une certaine configuration spatio-temporelle, mais sans aller plus loin dans la génération de comportements en conséquence. Cet ajout d'une représentation explicite du temps et de l'espace permet d'enrichir l'expressivité des normes, et in fine, les comportements qui en résultent.

D'autres propositions pour intégrer la notion d'espace, de coordonnées, et des formulations de quantités possibles sont également proposées dans [Rakotonirainy et al., 2011] qui sont systématiquement utilisées dans la simulation des socio-écosystèmes tels que [Aubert et al., 2010] ou, plus concrètement, une version à plus grande échelle du modèle IRIELA proposé. Cette notion spatio-temporelle permet notamment d'exprimer par exemple les normes suivantes :

- *"Il est interdit pour les non-villageois de cultiver dans la zone urbaine de Fianarantsoa"*
- *"Il est obligatoire pour les pêcheurs et les entreprises de pêche d'avoir une licence avant de pouvoir pêcher"*

Quant à la notion de quantités, elles permettraient d'enrichir encore plus l'expressivité de ces normes, par exemple en articulant les normes suivantes :

- *"Il est interdit pour pour les non-villageois de cultiver plus de 10Ha dans la zone urbaine de Fianarantsoa"*
- *Il est obligatoire pour les pêcheurs et les entreprises de pêche d'avoir une licence avant de pouvoir pêcher plus de 20Kg"*

En réponse, des comportements plus fins, qui prend en compte les normes normes et leur aspect spatio-temporels, les quantités envisageables dans ce type de système seraient :

- Lorsqu'il est interdit de faire telle activité dans telle zone, l'agent va alors se déplacer vers une autre zone où l'activité est permise;
- Lorsqu'il est interdit de faire telle activité durant telle période de temps, l'agent peut tout simplement attendre le temps favorable pour le faire;
- Si le taux de pêche souhaité est inférieur à la quantité de pêche autorisée, pêcher normalement sans licence, s'il est supérieur, contracter une licence pour pouvoir pêcher ou bien, trouver une zone qui permet de pêcher plus de 20kg sans avoir à acquérir une licence.

Pour cela, il nous faut comprendre la relation entre un plan et les informations spatio-temporelles : comment décrit-on que telle situation ou tel *pas* se situe dans un intervalle de temps ? qu'il se situe dans telle portion d'espace ? et comment on prend en compte ces quantités dans les planificateurs ?

Conclusion

Pour conclure, ce travail répond aux besoins des juristes et des agronomes qui se posent des questions sur l'efficacité des normes imposées par les institutions. Pour comprendre les effets de ces derniers, des modèles à base d'agents sont de plus en plus proposés, avec différentes architectures d'agent permettant de façonner le comportement des agents, y compris sous la contrainte des normes. Malgré l'existence de plusieurs architectures d'agent qui prennent en compte les normes, nous avons pu remarquer qu'elles se retrouvent limitées en raison d'une absence d'autonomie sociale : les agents choisissent parmi des comportements prédéfinis, ou bien, se trouvent forcés à suivre les normes sans l'autonomie d'accomplir ou non les objectifs de la norme.

Ceci empêche les juristes et les experts du domaine d'obtenir des SMA des éléments de réponse adéquats à leur(s) question(s) puisque : 1) les agents ne peuvent pas engendrer de nouveaux comportements même sous l'influence de nouvelles normes, et se limitent le plus souvent aux plans prédéfinis par le modélisateur ou le code ad-hoc de la plateforme de simulation; ce qui engendre 2) une explicabilité difficile sur les comportements des agents qui entraîne des doutes sur leur pertinence par rapport aux questions des experts du domaine; et in fine, 3) les modèles à base d'agents pourvus de normes sont difficilement interopérables et réutilisables en l'absence d'un standard pour permettre à l'agent de se construire un comportement normatif de zéro. Pour insuffler cette autonomie sociale des agents, nous proposons des stratégies plus explicites qui permettent aux agents de prendre en compte les normes dans la génération de comportement.

Le présent travail propose donc une architecture d'agent et son langage dédié pour que l'agent prenne en compte les normes, et engendre des comportements adéquats. Notre contribution se base sur un paradigme de planification à ordre partiel (POP), auquel nous avons ajouté la notion d'institutions et d'organisations qui imposent les normes telles qu'elles sont décrites par la grammaire institutionnelle de [Crawford & Ostrom, 1995]. Nous apportons une liste explicite et exhaustive de "comment les agents peuvent respecter des normes ?" que ce soit des actions ou des conditions du monde qu'on souhaite rendre obligatoire, interdite ou permises.

Nous proposons d'ajouter aux lacunes classiques du POP des lacunes normatives qui vont amener à des modifications du plan pour être résolues. Nous intégrons également une stratégie de contournement pour que les agents respectent les normes en rendant nécessairement vraie la négation d'une des conditions d'applicabilité de la norme, i.e. l'agent rend la norme inapplicable pour ne pas la violer.

Si ces contributions offrent une liste des stratégies possibles pour prendre en compte les normes dans la planification de l'agent, plusieurs points restent à aborder pour de futurs travaux : 1) les impacts de la quantité sur le comportement des agents; 2) les impacts des sanctions (récompenses, punitions) sur le comportement des agents; 3) l'intégration du contexte spatio-temporel dans la génération de comportements. D'autres améliorations en termes de performance sont également à prévoir puisque 1) A* n'est pas l'algorithme le plus efficace en termes de performance, même si ce travail en démontre la faisabilité ; 2) D'autres heuristiques plus spécialisées peuvent être employées ou paramétrées par le modélisateur; et 3) une étude sur l'ergonomie du langage et de la visualisation de l'arbre de recherche est nécessaire pour accompagner le modélisateur ainsi que les discussions des juristes et les autres experts du domaine.

Dans de futurs travaux, il est envisageable de relier la prise en compte des intervalles de temps et de l'espace proposée par [Raharivelo & Müller, 2018] pour les intégrer dans l'architecture proposée. En effet, ces travaux s'intéressent seulement au calcul de l'applicabilité des normes spatio-temporelles, mais leur intégration permet d'obtenir une nouvel arsenal d'opérateurs intéressants : 1) l'agent a conscience de où et quand une norme est applicable, il peut donc tout simplement décider d'attendre le moment opportun pour faire une action interdite, ou se rendre dans un autre endroit où l'action est permise; 2) l'agent dispose de la notion de deadline, puisque les actions obligatoires par exemple doivent s'accomplir entre le moment où la norme est applicable et le moment où elle ne l'est plus, ce qui peut aboutir à des comportements de priorisation qui pourrait grandement changer les plans des agents, et 3) ceci permettrait à l'agent de pouvoir "attendre" le bon moment, et/ou bien d'atteindre le bon "endroit" pour atteindre ses objectifs sans violer la norme. Plus loin, il serait envisageable de comprendre comment ces comportements produits sous l'influence des normes, conduisent aux modifications des normes elles-mêmes, voire à l'émergence de nouvelles normes.

Toutefois, de par son essence même, le présent travail hérite de la grande complexité inhérente aux espaces de recherche immenses engendrés par la planification automatisée. Pour mettre en pratique de tels concepts en tant que framework agents utilisables, il est indispensable de combiner à la fois ce type de planificateur avec des bibliothèques de plans prédéfinis, et des algorithmes d'apprentissage pour ne pas avoir à solliciter le planificateur à la moindre action habituelle. In fine, l'outil ainsi obtenu permettra de mieux répondre aux questions des experts du domaine sur les impacts des normes sur leur modèle, si elles sont efficaces ou non, et si elles ne sont pas efficaces, pourquoi ? et quelles normes proposer à la place ?

Références bibliographiques

[Adam & Gaudio, 2016] Adam, C. & Gaudou, B. (2016). BDI agents in social simulations: a survey. The Knowledge Engineering Review. 31. 207-238. 10.1017/S0269888916000096.

[Adam et al., 2011] Adam, C., Gaudou, B., Hickmott, S., & Scerri, D. (2011). Agents BDI et simulations sociales. Revue d'Intelligence Artificielle (RIA)-numéro spécial simulation multi-agents, 25(1).

[Alechina et al., 2012] . N. Alechina, M. Dastani, and B. Logan (2012). Programming norm-aware agents. In Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, pages 1057–1064, Richland, SC.

[Alechina et al., 2018] Alechina, N., Dastani, M., & Logan, B. (2018). Norm specification and verification in multi-agent systems. Journal of Applied Logics, 5(2), 457.

[Alzetta et al., 2020] Alzetta, F., Giorgini, P., Marinoni, M., & Calvaresi, D. (2020, June). RT-BDI: a real-time BDI model. In International Conference on Practical Applications of Agents and Multi-Agent Systems (pp. 16-29). Cham: Springer International Publishing.

[Amaral & Hubner, 2019] Amaral, C. J., & Hübner, J. F. (2019, May). Jacamo-web is on the fly: an interactive multi-agent system IDE. In International Workshop on Engineering Multi-Agent Systems (pp. 246-255). Cham: Springer International Publishing.

[Anderson, 1983] Anderson, J. A. (1983). Cognitive and psychological computation with neural models. IEEE transactions on systems, man, and cybernetics, (5), 799-815.

[Anderson & Lebiere, 2003] Anderson, J. R., & Lebiere, C. (2003). The Newell test for a theory of cognition. Behavioral and brain Sciences, 26(5), 587-601.

[Andrighetto & Conte, 2012] Andrighetto, G., & Conte, R. (2012). Cognitive dynamics of norm compliance. From norm adoption to flexible automated conformity. Artificial Intelligence and Law, 20, 359-381.

[Andrighetto et al., 2007] Andrighetto G, Campenn`ı M, Conte R, Paolucci M (2007) On the immergence of norms: a normative agent architecture. In: Proceedings of AAAI Symposium, Social and Organizational Aspects of Intelligence Washington DC

[Andrighetto et al., 2010] Andrighetto, G., Campenni, M., & Conte, R. (2010). Making the theory explicit: the EMIL-A architecture. EMergence In the Loop: simulating the two way dynamics of norm innovation, 77-88.

[Aubert, 2002] Aubert, S. (2002). La gestion patrimoniale des ressources forestières à Madagascar: limites et perspectives d'une révolution par le haut. Patrimonialiser la nature tropicale, Paris, IRD Éditions.

[Aubert et al., 2010] Aubert, S., Müller, J. P., & Ralihalizara, J. (2010). MIRANA: a socio-ecological model for assessing sustainability of community-based regulations.

[Aubert & Müller., 2013] Aubert, S., & Müller, J. P. (2013). Incorporating institutions, norms and territories in a generic model to simulate the management of renewable resources. Artificial Intelligence and Law, 21(1), 47-78.

[Axelrod, 1986] Axelrod, R. (1986). An evolutionary approach to norms. American political science review, 80(4), 1095-1111.

[Axelrod, 1997] Axelrod, R. (1997). Advancing the art of simulation in the social sciences. In Simulating social phenomena (pp. 21-40). Berlin, Heidelberg: Springer Berlin Heidelberg.

[Balbo et al., 2010] Balbo, F., Boissier, O., & Badeig, F. (2010). Spécification des modes d'interaction au sein d'organisations multi-agents. In JFSMA (pp. 141-150).

[Bacchus & Kabanza, 2000] Bacchus, F., & Kabanza, F. (2000). Using temporal logics to express search control knowledge for planning. Artificial intelligence, 116(1-2), 123-191.

[Barrett et al., 2015] Barrett, C., De Moura, L., & Fontaine, P. (2015). Proofs in satisfiability modulo theories. All about proofs, Proofs for all, 55(1), 23-44.

[Belouaer et al., 2012] Belouaer, L., Bouzid, M., & Mouaddib, A. I. (2012, May). Spatial Planning. In Journées Francophones sur la planification, la décision et l'apprentissage pour le contrôle des systèmes-JFPDA 2012 (pp. 13-p).

[Blum & Furst, 1997] Blum, A. L., & Furst, M. L. (1997). Fast planning through planning graph analysis. Artificial intelligence, 90(1-2), 281-300.

[Boella et al., 2006] Boella, G., Van Der Torre, L., & Verhagen, H. (2006). Introduction to normative multiagent systems. Computational & Mathematical Organization Theory, 12, 71-79.

[Boella et al., 2007] Boella, G., & Van Der Torre, L. (2007). Norm negotiation in multiagent systems. International Journal of Cooperative Information Systems, 16(01), 97-122.

[Boissier et al., 2013] Boissier, O., Bordini, R. H., Hübner, J. F., Ricci, A., & Santi, A. (2013). Multi-agent oriented programming with JaCaMo. Science of Computer Programming, 78(6), 747-761.

[Boissier et al., 2016] Boissier, O., Hübner, J. F., & Ricci, A. (2016). The jacamo framework. Social coordination frameworks for social technical systems, 125-151.

[Bousquet et al., 1998] Bousquet, F., Bakam, I., Proton, H., & Le Page, C. (1998, June). Cormas: common-pool resources and multi-agent systems. In International conference on industrial, engineering and other applications of applied intelligent systems (pp. 826-837). Berlin, Heidelberg: Springer Berlin Heidelberg.

[Bordini & Hübner, 2005] Bordini, R. H., & Hübner, J. F. (2005, June). BDI agent programming in AgentSpeak using Jason. In International workshop on computational logic in multi-agent systems (pp. 143-164). Berlin, Heidelberg: Springer Berlin Heidelberg.

[Bordini & Hübner, 2006] Bordini, R., & Hübner, J. F. (2006). An overview of Jason. Association for Logic Programming Newsletter, 19(3).

[Bourgais et al., 2016] Bourgais, M., Taillandier, P., & Vercouter, L. (2016, September). An agent architecture coupling cognition and emotions for simulation of complex systems. In Social simulation conference.

[Bousquet & Trébuil, 2005] Bousquet, F., & Trébuil, G. (2005). Introduction to companion modeling and multi-agent systems for integrated natural resource management in Asia.

Companion modeling and multi-agent systems for integrated natural resource management in Asia, 1-20.

[Bratman, 1987] Bratman, M. (1987). Intention, plans, and practical reason. The David Hume Series :Philosophy and Cognitivity Reissues. CSLI Publications.

[Bratman et al., 1987] Bratman, M. E., Israel, D. J., & Pollack, M. E. (1987). Toward an architecture for resource-bounded agents. SRI and Stanford Univ., Stanford, CA, Centre for the Study of Language Information, 87-104.

[Braziunas, 2003] Braziunas, D. (2003). POMDP solution methods. University of Toronto.

[Broersen et al., 2001] Broersen, J., Dastani, M., Hulstijn, J., Huang, Z., & Van Der Torre, L. (2001, May). The BOID architecture: conflicts between beliefs, obligations, intentions and desires. In Proceedings of the fifth international conference on Autonomous agents (pp. 9-16).

[Brooks, 1987] Brooks, R. (1987, March). A hardware retargetable distributed layered architecture for mobile robot control. In Proceedings. 1987 IEEE International Conference on Robotics and Automation (Vol. 4, pp. 106-110). IEEE.

[Canan & Birtürk, 2006] Canan, Ö., & Birtürk, A. (2006, June). Motion economy and planning. In Proceedings of the Sixteenth International Conference on International Conference on Automated Planning and Scheduling (pp. 366-369).

[Carley et al., 1998] Carley, K.M., Prietula, M.J., Lin, Z. (1998) : Design Versus Cognition: the Interaction of Agent Cognition and Organizational Design on Organizational Performance. Journal of Artificial Societies and Social Simulation 1

[Carmel & Markovitch, 1996] Carmel, D., & Markovitch, S. (1996, August). Learning models of intelligent agents. In AAAI/IAAI, Vol. 1 (pp. 62-67).

[Carter et al., 2005] Carter, D. P., Weible, C. M., Siddiki, S. N., Brett, J., & Chonaiew, S. M. (2015). Assessing policy divergence: how to investigate the differences between a law and a corresponding regulation. Public Administration, 93(1), 159-176.

[Castelfranchi, 2000] Castelfranchi, C. (2000, August). Founding agents'" autonomy" on dependence theory. In ECAI (Vol. 1, pp. 353-357).

[Castelfranchi et al., 2000] Castelfranchi, C., Dignum, F., Jonker, C. M., & Treur, J. (2000). Deliberative normative agents: Principles and architecture. In Intelligent Agents VI. Agent Theories, Architectures, and Languages: 6th International Workshop, ATAL'99, Orlando, Florida, USA, July 15-17, 1999. Proceedings 6 (pp. 364-378). Springer Berlin Heidelberg.

[Chapman, 1987] Chapman, D. (1987). Planning for conjunctive goals. Artificial intelligence, 32(3), 333-377.

[Chin et al., 2014] Chin, K. O., Gan, K. S., Alfred, R., Anthony, P., & Lukose, D. (2014). Agent architecture: An overview. Transactions on science and technology, 1(1), 18-35.

[Chong et al., 2007] Chong, H. Q., Tan, A. H., & Ng, G. W. (2007). Integrated cognitive architectures: a survey. Artificial Intelligence Review, 28, 103-130.

[Chong et al., 2011] Chong, L., Abbas, M. M., & Medina, A. (2011). Simulation of driver behavior with agent-based back-propagation neural network. Transportation Research Record, 2249(1), 44-51.

[Christensen et al., 2020] Christensen, K., Ma, Z., Demazeau, Y., & Jørgensen, B. N. (2020, March). Agent-based modeling for optimizing CO2 reduction in commercial greenhouse production with the implicit demand response. In 6th IEEJ International Workshop on Sensing, Actuation, Motion Control, and Optimization, SAMCON 2020.

[Cialdini & Trost, 1998] Cialdini, R. B., & Trost, M. R. (1998). Social influence: Social norms, conformity and compliance.

[Coleman, 1998] Coleman, J. S. (1990). Foundations of social theory. Harvard university press.

[Conte et al., 1998] Conte, R., Castelfranchi, C., & Dignum, F. (1998, July). Autonomous norm acceptance. In International Workshop on Agent Theories, Architectures, and Languages (pp. 99-112). Berlin, Heidelberg: Springer Berlin Heidelberg.

[Conte & Castelfranchi, 1995] Conte, R., & Castelfranchi, C. (1995). Cognitive And Social Action (1st ed.). Garland Science. Italian National Research Council, https://doi.org/10.4324/9780203783221

[Conte & Castelfranchi, 1999] Conte, R., & Castelfranchi, C. (1999). From conventions to prescriptions. Towards an integrated view of norms. Artificial intelligence and Law, 7, 323-340.

[Conte & Castelfranchi, 2006] Conte, R., & Castelfranchi, C. (2006). The mental path of norms. Ratio Juris, 19(4), 501-517.

[Crawford & Ostrom, 1995] Crawford, S. E., & Ostrom, E. (1995). A grammar of institutions. American political science review, 89(3), 582-600.

[Criado et al., 2010] Criado, N., Argente, E., & Botti, V. (2010, August). Rational strategies for norm compliance in the n-BDI proposal. In International Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems (pp. 1-20). Berlin, Heidelberg: Springer Berlin Heidelberg.

[d'Inverno et al, 2008] d'Inverno, M., Kinny, D., Luck, M., & Wooldridge, M. (1998). A formal specification of dMARS. In Intelligent Agents IV Agent Theories, Architectures, and Languages: 4th International Workshop, ATAL'97 Providence, Rhode Island, USA, July 24–26, 1997 Proceedings 4 (pp. 155-176). Springer Berlin Heidelberg.

[Dastani, 2008] Dastani, M. (2008). 2APL: a practical agent programming language. Autonomous agents and multi-agent systems, 16, 214-248.

[Davidsson, 2002] Davidsson, P. (2002). Agent based social simulation: A computer science view. Journal of artificial societies and social simulation, 5(1).

[de Campos, 2012 et al., 2012] de Campos, G. A., Freire, E. S., & Cortés, M. I. (2012). Norm-based behavior modification in reflex agents. In Proceedings on the International Conference on Artificial Intelligence (ICAI) (p. 1). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

[De Mour & Bjorner, 2011] De Moura, L., & Bjørner, N. (2011). Satisfiability modulo theories: introduction and applications. Communications of the ACM, 54(9), 69-77.

[de Silva et al., 2009] De Silva, L., Sardina, S., & Padgham, L. (2009). First principles planning in BDI systems. In Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems (pp. 1105-1112). International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS).

[Dean & Boddy, 1988] Dean, T. L., & Boddy, M. S. (1988, August). An Analysis of Time-Dependent Planning. In AAAI (Vol. 88, pp. 49-54).

[Dembytskyi & Dorogyy, 2017] Dembytskyi, A., & Dorogyy, Y. (2017, October). Agent-based modeling of the human behavior with genetic algorithm. In 2017 4th International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T) (pp. 87-92). IEEE.

[Dignum, 2018] Dignum, F. (2018). Interactions as social practices: towards a formalization. arXiv preprint arXiv:1809.08751.

[Dignum, 2021] Dignum, F. (2021). Foundations of social simulations for crisis situations. In Social simulation for a crisis: Results and lessons from simulating the COVID-19 crisis (pp. 15-37). Cham: Springer International Publishing.

[Dignum & Dignum, 2001] Dignum, V., & Dignum, F. (2001). Modelling agent societies: Co-ordination frameworks and institutions. In Progress in Artificial Intelligence: Knowledge Extraction, Multi-agent Systems, Logic Programming, and Constraint Solving 10th Portuguese Conference on Artificial Intelligence, EPIA 2001 Porto, Portugal, December 17–20, 2001 Proceedings 10 (pp. 191-204). Springer Berlin Heidelberg.

[Ding et al., 2015] Ding, D., Bennett, D., & Secchi, S. (2015). Investigating impacts of alternative crop market scenarios on land use change with an agent-based model. Land, 4(4), 1110-1137.

[dos Santos Neto et al., 2012] dos Santos Neto, B. F., da Silva, V. T., & de Lucena, C. J. (2012, October). An architectural model for autonomous normative agents. In Brazilian Symposium on Artificial Intelligence (pp. 152-161). Berlin, Heidelberg: Springer Berlin Heidelberg.

[Drogoul et al., 2002] Drogoul, A., Vanbergue, D., & Meurisse, T. (2002, July). Multi-agent based simulation: Where are the agents?. In International Workshop on Multi-Agent Systems and Agent-Based Simulation (pp. 1-15). Berlin, Heidelberg: Springer Berlin Heidelberg.

[Erol et al., 1992] Erol, K., Nau, D. S., & Subrahmanian, V. S. (1992, January). When is planning decidable?. In Artificial Intelligence Planning Systems (pp. 222-227). Morgan Kaufmann.

[Erol et al., 1994] Erol, K., Hendler, J., & Nau, D. S. (1994, July). HTN planning: Complexity and expressivity. In AAAI (Vol. 94, pp. 1123-1128).

[Esteva et al., 2004] Esteva, M., Rosell, B., Rodriguez-Aguilar, J. A., & Arcos, J. L. (2004, July). AMELI: An agent-based middleware for electronic institutions. In AAMAS (Vol. 4, pp. 236-243).

[Fagundes et al., 2016] Fagundes, M. S., Ossowski, S., Cerquides, J., & Noriega, P. (2016). Design and evaluation of norm-aware agents based on Normative Markov Decision Processes. International Journal of Approximate Reasoning, 78, 33-61.

[Ferber, 1994] Ferber, J. (1994). Coopération réactive et émergence. Intellectica, 19(2), 19-52.

[Ferber et al., 2003] Ferber, J., Gutknecht, O., & Michel, F. (2003, April). Agent/group/roles: Simulating with organizations. In Fourth International Workshop on Agent-Based Simulation (ABS03).

[Figueiredo et al., 2010] da Silva Figueiredo, K., Torres da Silva, V., & de Oliveira Braga, C. (2010, August). Modeling norms in multi-agent systems with NormML. In International Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems (pp. 39-57). Berlin, Heidelberg: Springer Berlin Heidelberg.

[Fikes & Nilsson, 1971] Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. Artificial intelligence, 2(3-4), 189-208.

[Fornara et al., 2013] Fornara, N., Cardoso, H. L., Noriega, P., Oliveira, E., Tampitsikas, C., & Schumacher, M. I. (2013). Modelling agent institutions. Agreement technologies, 277-307.

[Franklin & Patterson, 2006] Franklin, S., & Patterson, F. (2006). The LIDA architecture: Adding new modes of learning to an intelligent, autonomous, software agent.

[Frantz, 2015] Frantz, C. K. (2015). Agent-based institutional modelling: Novel techniques for deriving structure from behaviour. University of Otago (New Zealand).

[Frantz et al., 2015] Frantz, C. K., Purvis, M. K., Savarimuthu, B. T. R., & Nowostawski, M. (2014). Modelling dynamic normative understanding in agent societies. In PRIMA 2014: Principles and Practice of Multi-Agent Systems: 17th International Conference, Gold Coast, QLD Australia, December 1-5, 2014. Proceedings 17 (pp. 294-310). Springer International Publishing.

[Frantz & Siddiki, 2020] Frantz, C. K., & Siddiki, S. N. (2020). Institutional grammar 2.0 codebook. arXiv preprint arXiv:2008.08937.

[Fox & Long, 2003] Fox, M., & Long, D. (2003). PDDL2. 1: An extension to PDDL for expressing temporal planning domains. Journal of artificial intelligence research, 20, 61-124.

[Georgeff & Ingrand, 1989] Georgeff, M. P., & Ingrand, F. (1989). Decision-making in an embedded reasoning system. In International joint conference on artificial intelligence.

[Gérévini & Long, 2005] Gerevini, A., & Long, D. (2005). Plan constraints and preferences in PDDL3. Technical Report 2005-08-07, Department of Electronics for Automation, University of Brescia, Brescia, Italy.

[Gil-Quijano et al., 2007] Gil-Quijano, J., Piron, M. & Drogoul, A. (2007). Mechanisms of automated formation and evolution of social-groups: a multi-agent system to model the intra-urban mobilities of Bogota city. In Social Simulation : Technologies, Advances and New Discoveries, Chapter 12, Edmonds B., Hernandez C. & Troitzsch K. (eds). Idea Group Inc.,151–168

[Ghallab et al., 1998] Ghallab, M., Howe, A., Knoblock, C., McDermott, I. D., Ram, A., Veloso, M., ... & Sun, Y. (1998). Pddl| the planning domain definition language. Technical Report, Tech. Rep.

[Ghallab et al., 2004] Ghallab, M., Nau, D., & Traverso, P. (2004). Automated Planning: theory and practice. Elsevier.

[Ghorbani & Bravo, 2016] Ghorbani, A., & Bravo, G. (2016). Managing the commons: a simple model of the emergence of institutions through collective action. International Journal of the Commons, 10(1), 200-219.

[Ghorbani et al., 2013] Ghorbani, A., Dignum, V., Bots, P., & Dijkema, G. (2013). MAIA: a framework for developing agent-based social simulations. JASSS-The Journal of Artificial Societies and Social Simulation, 16(2), 9.

[Ghorbani et al., 2015] Ghorbani, A., Dijkema, G. P., & Schrauwen, N. (2015). Structuring qualitative data for agent-based modelling. JASSS-The Journal of Artificial Societies and Social Simulation, 18(1), 2.

[Hannoun et al., 1999] Hannoun, M., Boissier, O., Sichman, J. S., & Sayettat, C. (1999). Moïse: Un modèle organisationnel pour la conception de systèmes multi-agents. JFIADSMA, 99, 105-118.

[Heidari et al., 2020] Heidari, S., Jensen, M., & Dignum, F. (2020). Simulations with values. In Advances in Social Simulation: Looking in the Mirror (pp. 201-215). Springer International Publishing.

[Helbing et al., 2000] Helbing, D., Farkas, I., & Vicsek, T. (2000). Simulating dynamical features of escape panic. Nature, 407(6803), 487-490.

[Hendler et al., 1990] Hendler, J. A., Tate, A., & Drummond, M. (1990). AI planning: Systems and techniques. AI magazine, 11(2), 61-61.

[Hindriks et al., 2010] Hindriks, K. V., van Riemsdijk, M. B., Behrens, T., Korstanje, R., Kraaijenbrink, N., Pasman, W. & de Rijk, L. (2010). Unreal goal bots. connecting agents to complex dynamic environments. In AGS 2010.

[Hodgson, 2006] Hodgson, G. M. (2006). What are institutions?. Journal of economic issues, 40(1), 1-25.

[Holland, 1975] HOLLAND, J. H. 1975. Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor, MI.

[Hollander & Wu, 2011] Hollander, C. D., & Wu, A. S. (2011). The current state of normative agent-based systems. Journal of Artificial Societies and Social Simulation, 14(2), 6.

[Horvitz, 1987] Horvitz, E. J. (1987). Reasoning about beliefs and actions under computational resource constraints. In Proceedings of the 1987 Workshop on Uncertainty in Artificial Intelligence, Seattle, Washington,

[Hubner et al., 2002] Hübner, J. F., Sichman, J. S., & Boissier, O. (2002, July). MOISE+ towards a structural, functional, and deontic model for MAS organization. In Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1 (pp. 501-502).

[Jackson et al., 2021] Jackson, R. B., Li, S., Banisetty, S. B., Siva, S., Zhang, H., Dantam, N., & Williams, T. (2021, September). An integrated approach to context-sensitive moral cognition in robot cognitive architectures. In 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 1911-1918). IEEE.

[Jennings & Wooldridge, 1998] Jennings, N. R., & Wooldridge, M. (1998). Applications of intelligent agents. Agent technology: foundations, applications, and markets, 3-28.

[Kammler et al., 2021] Kammler, C., Dignum, F., Wijermans, N., & Lindgren, H. (2021, May). Changing perspectives: adaptable interpretations of norms for agents. In International Workshop on Multi-Agent Systems and Agent-Based Simulation (pp. 139-152). Cham: Springer International Publishing.

[Kammler et al., 2022a] Kammler, C., Mellema, R., & Dignum, F. (2022, May). Agents dealing with norms and regulations. In International Workshop on Multi-Agent Systems and Agent-Based Simulation (pp. 134-146). Cham: Springer International Publishing.

[Kammler et al., 2022b] Kammler, C., Dignum, F., & Wijermans, N. (2022, September). Utilizing the full potential of norms for the agent's decision process. In Conference of the European Social Simulation Association (pp. 193-205). Cham: Springer Nature Switzerland.

[Kautz & Selman, 1999] Kautz, H., & Selman, B. (1999, June). Unifying SAT-based and graph-based planning. In IJCAI (Vol. 99, pp. 318-325).

[Kollingbaum & Norman, 2003] Kollingbaum, M. J., & Norman, T. J. (2003, July). Norm adoption in the NoA agent architecture. In Proceedings of the second international joint conference on Autonomous agents and multiagent systems (pp. 1038-1039).

[Kammler et al., 2022] Kammler, C., Dignum, F., & Wijermans, N. (2022, September). Utilizing the full potential of norms for the agent's decision process. In Conference of the European Social Simulation Association (pp. 193-205). Cham: Springer Nature Switzerland.

[Kokinov, 1994] Kokinov, B. N. (1994, August). The DUAL Cognitive Architecture: A Hybrid Multi-Agent Approach. In ECAI (pp. 203-207).

[Krzisch & Meneguzzi, 2017] Krzisch, G., & Meneguzzi, F. R. (2017). Planning in a Normative System. In Proceedings of the 2017 International Workshop on Coordination, Organisations, Institutions and Norms (COIN), 2017, Brasil..

[Kvarnström, 2011] Kvarnström, J. (2011, March). Planning for loosely coupled agents using partial order forward-chaining. In Proceedings of the International Conference on Automated Planning and Scheduling (Vol. 21, pp. 138-145).

[Laird, 1986] Laird, J. E. (1986). SOAR User's Manual. INSTITUTION Xerox Corp., Palo Alto, CA. Palo Alto Research Center. SPONS AGENCY Office cf Naval Research, Arlington, Va. Personnel. CONTRACT, 14, 82C-0067.

[Laird et al., 1987] Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. Artificial intelligence, 33(1), 1-64.

[Laird et al., 1990] Laird, J. E., Newell, A., & McCarl, R. (1990). A Preliminary Analysis of the Soar Architecture as a Basis for General Intelligence*,†,‡ Paul S. Rosenbloom $ Information Sciences Institute University of Southern California. In Innovative Approaches to Planning, Scheduling and Control: Proceedings of the 1990 DARPA Workshop (p. 468). Morgan Kaufmann.

[Lawrence, 1976] D. G. Lawrence (1976), “Procedural norms and tolerance: a reassessment,” The American PoliticalScience Review, vol. 70, no. 1, pp. 80–100.

[Lebiere & Anderson, 1993] Lebiere, C., & R Anderson, J. (1993). A connectionist implementation of the ACT-R production system.

[Lee et al., 2014] Lee, J., Padget, J. A., Logan, B., Dybalova, D., & Alechina, N. (2014, May). Run-time norm compliance in BDI agents. In AAMAS (pp. 1581-1582).

[Likhachev et al., 2005] Likhachev, M., Ferguson, D. I., Gordon, G. J., Stentz, A., & Thrun, S. (2005, June). Anytime dynamic A*: An anytime, replanning algorithm. In ICAPS (Vol. 5, pp. 262-271).

[Lopez, 2003] López y López, F. (2003). Social Power and Norms: Impact on agent behaviour (Doctoral dissertation, University of Southampton).

[Lopez & Márquez, 2004] Lopez, F. L. Y., & Márquez, A. A. (2004, September). An architecture for autonomous normative agents. In Proceedings of the Fifth Mexican International Conference in Computer Science, 2004. ENC 2004. (pp. 96-103). IEEE.

[Lotzmann et al., 2009] Lotzmann, U., Möhring, M., & Troitzsch, K. G. (2009). Simulating the emergence and innovation of norms. PerAda Newsroom.

[Luck et al., 2013] Luck, M., Mahmoud, S., Meneguzzi, F., Kollingbaum, M., Norman, T. J., Criado, N., & Fagundes, M. S. (2013). Normative agents. Agreement technologies, 209-220.

[Maes, 1991] Maes, P. (1991). The agent network architecture (ANA). Acm sigart bulletin, 2(4), 115-120.

[Mahmoud et al., 2014] Mahmoud, M. A., Ahmad, M. S., Mohd Yusoff, M. Z., & Mustapha, A. (2014). A review of norms and normative multiagent systems. The Scientific World Journal, 2014.

[Maia & Sichman, 2020] Maia, A. V., & Sichman, J. S. (2020). Representing planning autonomy in agent organizational models. Theoretical Computer Science, 805, 92-108.

[McAllester & Rosenblatt, 1991] McAllester, D., & Rosenblatt, D. (1991). Systematic nonlinear planning.

[Meneguzzi & De Silva, 2015] Meneguzzi, F., & De Silva, L. (2015). Planning in BDI agents: a survey of the integration of planning algorithms and agent reasoning. The Knowledge Engineering Review, 30(1), 1-44.

[Meneguzzi & Luck, 2008] Meneguzzi, F., & Luck, M. (2007, May). Composing high-level plans for declarative agent programming. In International Workshop on Declarative Agent Languages and Technologies (pp. 69-85). Berlin, Heidelberg: Springer Berlin Heidelberg.

[Meneguzzi & Luck, 2009] Meneguzzi, F. R., & Luck, M. (2009, May). Norm-based behavior modification in BDI agents. In AAMAS (1) (pp. 177-184).

[Meneguzzi et al., 2004] Meneguzzi, F. R., Zorzo, A. F., & da Costa Móra, M. (2004, March). Propositional planning in BDI agents. In Proceedings of the 2004 ACM symposium on Applied computing (pp. 58-63).

[Meneguzzi et al., 2015] Meneguzzi, F., Rodrigues, O., Oren, N., Vasconcelos, W. W., & Luck, M. (2015). BDI reasoning with normative considerations. Engineering Applications of Artificial Intelligence, 43, 127-146.

[Mercuur et al., 2019] Mercuur, R., Dignum, V., & Jonker, C. (2017). Using Values and Norms to Model Realistic Social Agents. In 29th Benelux Conference on Artificial Intelligence November 8–9, 2017, Groningen (p. 357).

[Mintzberg, 1989] Mintzberg, H. (1989). The structuring of organizations (pp. 322-352). Macmillan Education UK.

[Müller, 2004] Müller, J. P. (2004). The mimosa generic modelling and simulation platform : The case of multi-agent systems. In : Proceedings of the 5th Workshop on Agent-Based Simulation, Lisbon, 2004. SCS. s.l. : s.n., 77-86. International Workshop on Agent-Based Simulation. 5, Lisbonne, Portugal, 3 Mai 2004/5 Mai 2004.

[Naveh & Sun, 2006] Naveh, I., & Sun, R. (2006). Simulating a simple case of organizational decision making. Cognition and Multi-Agent Interaction, 124.

[Neumann, 2010] Neumann, M. (2010). A classification of normative architectures. In Simulating Interacting Agents and Social Phenomena: The Second World Congress (pp. 3-18). Springer Japan.

[Newell, 1990] Newell, A. (1994). Unified Theories of Cognition, Harvard University Press; Reprint edition, ISBN 0-674-92101-1.

[Nguyen & Kambhampati, 2001] Nguyen, X., & Kambhampati, S. (2001, August). Reviving partial order planning. In IJCAI (Vol. 1, pp. 459-464).

[Norman & Reed, 2000] Norman, T. J., & Reed, C. (2000, July). Delegation and responsibility. In International Workshop on Agent Theories, Architectures, and Languages (pp. 136-149). Berlin, Heidelberg: Springer Berlin Heidelberg.

[North, 1990] North, D. C. (1990). Institutions, institutional change and economic performance. Cambridge university press.

[North, 1991] North, D. C. (1991). Institutions, ideology, and economic performance. Cato J., 11, 477.

[North, 1994] North, D. C. (1994). Institutions matter. Economic History, 9411004, 361.

[Novo & Garrido, 2014] Novo, P., & Garrido, A. (2014). From policy design to implementation: an institutional analysis of the new Nicaraguan Water Law. Water Policy, 16(6), 1009-1030.

[Oren et al., 2011] Oren, N., Vasconcelos, W., Meneguzzi, F., & Luck, M. (2011). Acting on norm constrained plans. In Computational Logic in Multi-Agent Systems: 12th International Workshop, CLIMA XII, Barcelona, Spain, July 17-18, 2011. Proceedings 12 (pp. 347-363). Springer Berlin Heidelberg.

[Orkin, 2006] Orkin, J. (2006, March). Three states and a plan: the AI of FEAR. In Game developers conference (Vol. 2006, p. 4). SanJose, California: CMP Game Group.

[Ostrom, 1990] Ostrom, E. (1990). Governing the Commons: The Evolution of Institutions for Collective Action. Cambridge University Press.

[Ostrom, 2005a] Ostrom O., 2005, Understanding institutional diversity, Princeton, Princeton University Press.

[Ostrom, 2005b] Ostrom, E. (2005). "Policies That Crowd out Reciprocity and Collective Action", Moral Sentiments and Material Interests: The Foundations of Cooperation in Economic Life, Herbert Gintis, Samuel Bowles, Robert Boyd, Ernst Feh.

[Ostrom, 2009] Ostrom, E. (2009). Building trust to solve commons dilemmas: Taking small steps to test an evolving theory of collective action (pp. 207-228). Springer Berlin Heidelberg.

[Ostrom, 2019] Ostrom, Elinor. "Institutional rational choice: An assessment of the institutional analysis and development framework." Theories of the Policy Process, Second Edition. Routledge, 2019. 21-64.

[Ostrom & Crawford, 1995] Crawford, S. E., & Ostrom, E. (1995). A grammar of institutions. American political science review, 89(3), 582-600.

[Panagiotidi & Vazquez-Salceda, 2011] Panagiotidi, S., & Vázquez-Salceda, J. (2011, August). Towards practical normative agents: A framework and an implementation for norm-aware planning. In International Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems (pp. 93-109). Berlin, Heidelberg: Springer Berlin Heidelberg.

[Panagiotidi et al., 2013] Panagiotidi, S., Vázquez-Salceda, J., & Dignum, F. (2013). Reasoning over norm compliance via planning. In Coordination, Organizations, Institutions, and Norms in Agent Systems VIII: 14th International Workshop, COIN 2012, Held Co-located with AAMAS 2012, Valencia, Spain, June 5, 2012, Revised Selected Papers 14 (pp. 35-52). Springer Berlin Heidelberg.

[Raharivelo & Müller, 2018] Raharivelo S. O., Müller J. P. (2018). Un modèle de norme intégrant les conditions spatiotemporelles. In : JFSMA 2018. Systèmes multi-agents : Distribution et décentralisation. Picard Guillaume (ed.), Lang Christophe (ed.), Marilleau Nicolas (ed.). Toulouse : Cépaduès, 117-126. ISBN 978-2-36493-675-1 Journées francophones sur les systèmes multi-agents (JFSMA 2018). 26, Métabief, France, 10 Octobre 2018/12 Octobre 2018.

[Rakotonirainy, 2016] Rakotonirainy, H. L. (2016). Exploration sémantique des modèles socio-environnementaux (Doctoral dissertation, Université de Fianarantsoa; UMR GREEN-CIRAD).

[Rakotonirainy et al., 2011] Towards a description logic for scientific modeling. In : International Conference on Knowledge Engineering and Ontology Development (KEOD). 3rd International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K), Paris, France, 26-29 October, 2011. s.l. : s.n., 6 p. International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management. 3, Paris, France, 26th October 2011/29th October 2011.

[Rakotonirainy et al., 2015a] Rakotonirainy, H. L., Müller, J. P., & Ramamonjisoa, B. O. (2015). A generic approach for initializing complex models. In : African University of Science and Technology (AUST) International Conference in Technology (AUSTECH) 2015. s.l. : s.n., 8 p. African University of Science and Technology (AUST) International Conference in Technology (AUSTECH) 2015, Abuja, Nigeria, 12th October2015/13th October 2015.

[Rakotonirainy et al., 2015b] Rakotonirainy, H. L., Müller, J. P., & Ramamonjisoa, B. O. (2015). Ingénierie Dirigée par les Modèles (IDM) appliquée à l'observation des modèles socio-environnementaux. In : 4th Conférence en IngénieriE du Logiciel CIEL2015. Bordeaux : INRA-INRIA, 6 p. Conférence en IngénieriE du Logiciel CIEL2015. 4, Bordeaux, France, 9th Juine 2015/11th June 2015.

[Ramamonjisoa et al., 2012] Ramamonjisoa, B., Rakoto Ramiarantsoa, H., & Casse, T. (2012). La Loi Gelose et le transfert de gestion des ressources naturelles à Madagascar. Les Cahiers d'Outre-Mer. Revue de géographie de Bordeaux, 65(257), 5-10.

[Ramarozaka et al., 2021] Ramarozaka, T., Müller, J. P., & Rakotonirainy, H. L (2021, August). Accounting norms in agent decision process to assess norm efficiency. In IJCAI 2021 DC.

[Ramarozaka et al., 2022] Ramarozaka, T., Müller, J. P., & Rakotonirainy, H. L. (2022, September). Extending Partial-Order Planning to Account for Norms in Agent Behavior. In Conference of the European Social Simulation Association (pp. 125-137). Cham: Springer Nature Switzerland.

[Rao & Georgeff, 1991] Rao, A. S. & Georgeff, M. P. (1991). Modeling rational agents within a BDI-architecture. In Fikes, R. and Sandewall, E., editors, Proceedings of Knowledge Representation and Reasoning (KR&R-91), 473-484. Morgan Kaufmann Publishers: San Mateo, CA.

[Rao & Georgeff, 1995] A. S. Rao and M. P. Georgeff (1995). BDI-agents: from theory to practice. In Proceedings of the First Intl. Conference on Multiagent Systems, San Francisco.

[Ricci et al., 2011] Ricci, A., Piunti, M., Viroli, M., & Omicini, A. (2009). Environment programming in CArtAgO. Multi-agent programming: Languages, tools and applications, 259-288.

[Russell & Norvig, 1995] Russell, S., & Norvig, P. (1995). A modern, agent-oriented approach to introductory artificial intelligence. Acm Sigart Bulletin, 6(2), 24-26.

[Sacerdoti, 1975] Sacerdoti, E. D. (1975). The nonlinear nature of plans. Stanford Research Institute.

[Sarrasin, 2009] Sarrasin, B. (2009). La Gestion Locale Sécurisée (GELOSE): L'expérience malgache de gestion décentralisée des ressources naturelles. Études caribéennes, (12).

[Sardina et al., 2016] Sardina, S., De Silva, L., & Padgham, L. (2006, May). Hierarchical planning in BDI agent programming languages: A formal approach. In Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems (pp. 1001-1008).

[Savarimuthu, 2011] Savarimuthu, B. T. R. (2011). Mechanisms for norm emergence and norm identification in multi-agent societies [Ph.D. thesis], University of Otago, Dunedin, New Zealand, 2011.

[Schut et al., 2002] Schut, M., Wooldridge, M., & Parsons, S. (2002). On partially observable MDPs and BDI models. In Foundations and Applications of Multi-Agent Systems: UKMAS Workshops 1996–2000 Selected Papers (pp. 243-259). Springer Berlin Heidelberg.

[Schwartz, 2012] Schwartz, S. H. (2012). An overview of the Schwartz theory of basic values. Online readings in Psychology and Culture, 2(1), 11.

[Searle, 1969] Searle, J.R. (1969). How to derive ‘ought’ from ‘is’. In: Hudson, W.D. (eds) The Is-Ought Question. Controversies in Philosophy. Palgrave Macmillan, London. https://doi.org/10.1007/978-1-349-15336-7_13

[Searle, 2005] Searle, J. R. (2005). What is an institution?. Journal of institutional economics, 1(1), 1-22.

[Shoham & Leyton-Brown, 2008] Shoham, Y., & Leyton-Brown, K. (2008). Multiagent systems: Algorithmic, game-theoretic, and logical foundations. Cambridge University Press.

[Siddiki & Frantz, 2023] Siddiki, S., & Frantz, C. K. (2023). Understanding regulation using the Institutional Grammar 2.0. Regulation & Governance.

[Siddiki et al., 2022] Siddiki, S., Heikkila, T., Weible, C. M., Pacheco‐Vega, R., Carter, D., Curley, C., ... & Bennett, A. (2022). Institutional analysis with the institutional grammar. Policy Studies Journal, 50(2), 315-339.

[Smajgl et al., 2008] Smajgl, A., Izquierdo, L. R., & Huigen, M. (2008). Modeling endogenous rule changes in an institutional context: The adico sequence. Advances in Complex Systems, 11(02), 199-215.

[Stollberg & Rhomberg, 2006] Stollberg M, Rhomberg F (2006) Survey on goal-driven architectures, Tech. rep., DERI, Austria

[Sun et al., 2001] Sun, R., Merrill, E., & Peterson, T. (2001). From implicit skills to explicit knowledge: A bottom‐up model of skill learning. Cognitive science, 25(2), 203-244.

[Sun, 1994] Sun, R. (1994). Integrating rules and connectionism for robust commonsense reasoning. New York: John Wiley and Sons.

[Sun, 2007] Sun, R. (2007). The importance of cognitive architectures: An analysis based on CLARION. Journal of Experimental & Theoretical Artificial Intelligence, 19(2), 159-193.

[Sun & Alexandre, 1997] Sun, R., & Alexandre, F., (1997), Connectionist Symbolic Integration, Lawrence Erlbaum Associates, Hillsdale, NJ.

[Sun et al., 2001] Sun, R., Merrill, E., and Peterson, T. (2001). From implicit skills to explicit knowledge: A bottom-up model of skill learning. Cognitive Science. 25 (2), 203-244.

[Taillandier et al., 2012] Taillandier, P., Vo, D. A., Amouroux, E., & Drogoul, A. (2012). GAMA: a simulation platform that integrates geographical information data, agent-based modeling and multi-scale control. In Principles and Practice of Multi-Agent Systems: 13th International Conference, PRIMA 2010, Kolkata, India, November 12-15, 2010, Revised Selected Papers 13 (pp. 242-258). Springer Berlin Heidelberg.

[Tisue & Wilensky] Tisue, S., & Wilensky, U. (2004, May). Netlogo: A simple environment for modeling complexity. In International conference on complex systems (Vol. 21, pp. 16-21).

[Tomic et al., 2018] Tomic, S., Pecora, F., & Saffiotti, A. (2018). Norms, institutions, and robots. arXiv preprint arXiv:1807.11456.

[Tradli et al., 2022] Traldi, A., Bruschetti, F., Robol, M., Roveri, M., & Giorgini, P. (2022). Real-time BDI agents: a model and its implementation. arXiv preprint arXiv:2205.00979.

[Ulmann-Margalit, 1977] Ulmann-Margalit, E. (1977). Coordination Norms. Edna Ullmann-Margalit, The Emergence of Norms, Oxford University Press, Oxford, 74-114.

[Viana et al., 2015] Viana, M. L., Alencar, P. S., Guimarães, E. T., Cunha, F. J., Cowan, D. D., & de Lucena, C. J. P. (2015, July). JSAN: A Framework to Implement Normative Agents. In SEKE (pp. 660-665).

[Walczak et al., 2007] Walczak, A., Braubach, L., Pokahr, A., & Lamersdorf, W. (2007). Augmenting BDI agents with deliberative planning techniques. In Programming Multi-Agent Systems: 4th International Workshop, ProMAS 2006, Hakodate, Japan, May 9, 2006, Revised and Invited Papers 4 (pp. 113-127). Springer Berlin Heidelberg.

[Watkins & Westphal, 2016] Watkins, C., & Westphal, L. M. (2016). People don't talk in institutional statements: A methodological case study of the institutional analysis and development framework. Policy Studies Journal, 44(S1), S98-S122.

[Wein & Labiosa, 2013] Wein, A., & Labiosa, W. (2013). Serious games experiment toward agent-based simulation (No. 2013-1152). US Geological Survey.

[Weld, 1999] Weld, D. S. (1999). Recent advances in AI planning. AI magazine, 20(2), 93-93.

[Wirth, 1977] Wirth, N. (1977). What can we do about the unnecessary diversity of notation for syntactic definitions?. Communications of the ACM, 20(11), 822-823.

[Wooldridge & Jennings, 1995] Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. The knowledge engineering review, 10(2), 115-152.

[Wray & Jones, 2005] Wray, R. E., & Jones, R. M. (2005). An introduction to Soar as an agent architecture. Cognition and multi-agent interaction: From cognitive modeling to social simulation, 53-78.

[Ye et al., 2018] Ye, P., Wang, T., & Wang, F. Y. (2018). A survey of cognitive architectures in the past 20 years. IEEE transactions on cybernetics, 48(12), 3280-3290.

[Young, 2007] Young, H. (2007). Social Norms. Department of Economics (University of Oxford).

Références webographiques

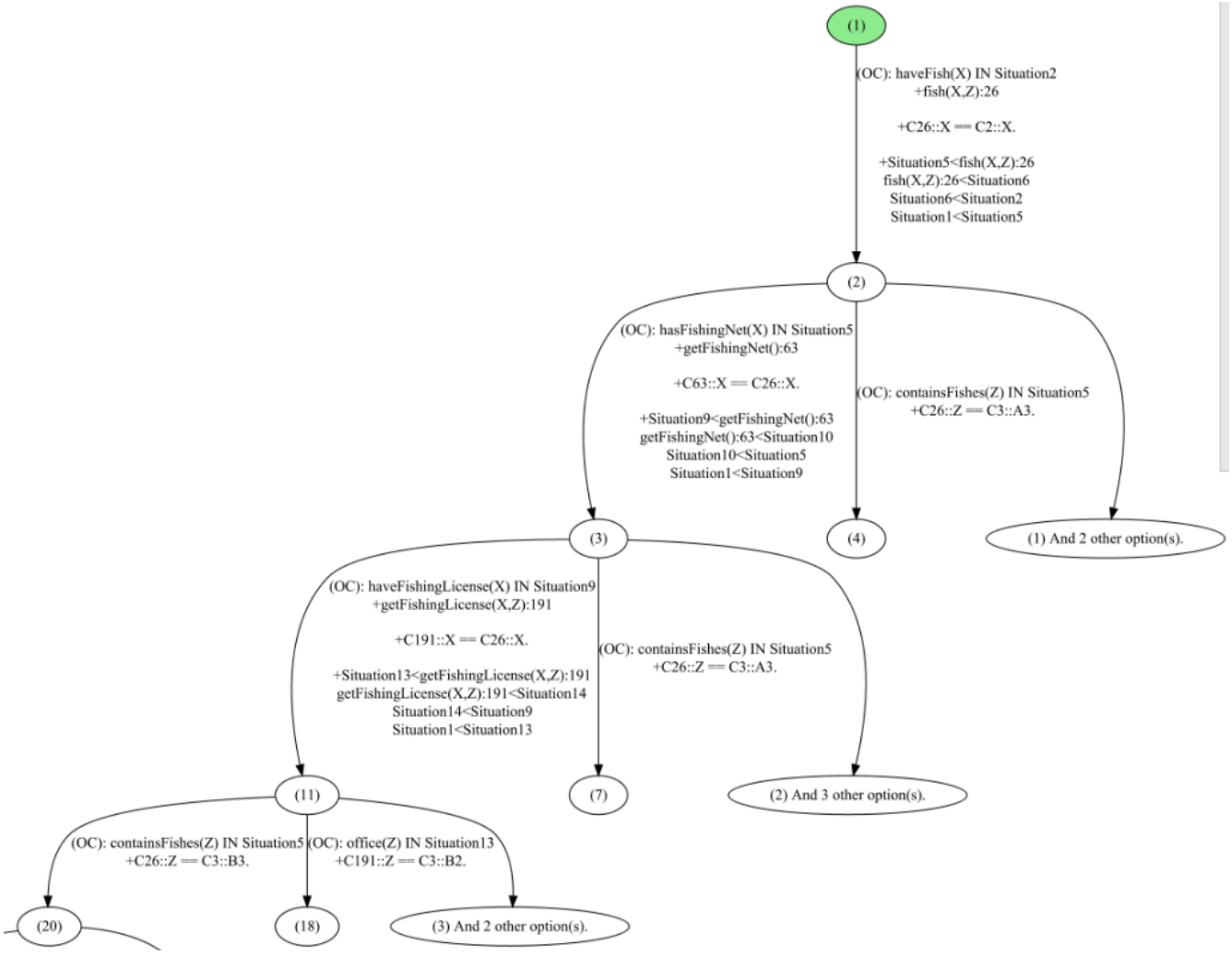
Lahodium, Y., “The evolution of agents controlled by a neural network. Disponible sur https://habrahabr.ru/post/168067 , consulté le 24 Avril 2024.

Kleiner, A., & Nebel, B. (2008). Introduction to Multi-Agent Programming. Disponible sur http://gki.informatik.uni-freiburg.de/teaching/ws0809/map/mas_lect3.pdf WS 2008/2009. Consulté le 15 Avril 2024.

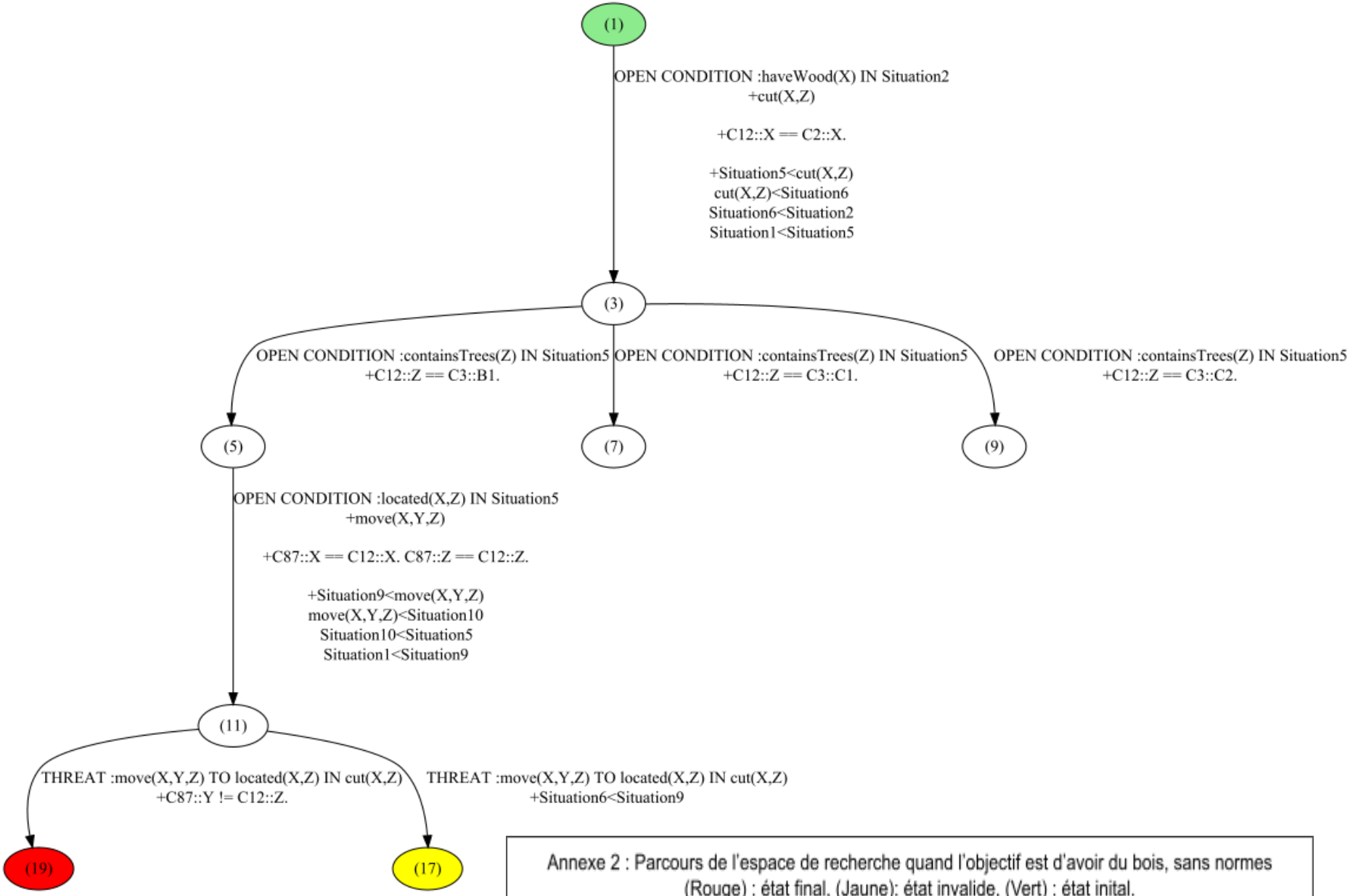
Ulieru, M.(2010). Intro to Foundations of Artificial Intelligence. Lecture 5. Disponible sur http://www.theimpactinstitute.org/Teaching/CS6705/Lecture5.pdf Winter 2010. Consulté le 15 Avril 2024.

Annexes

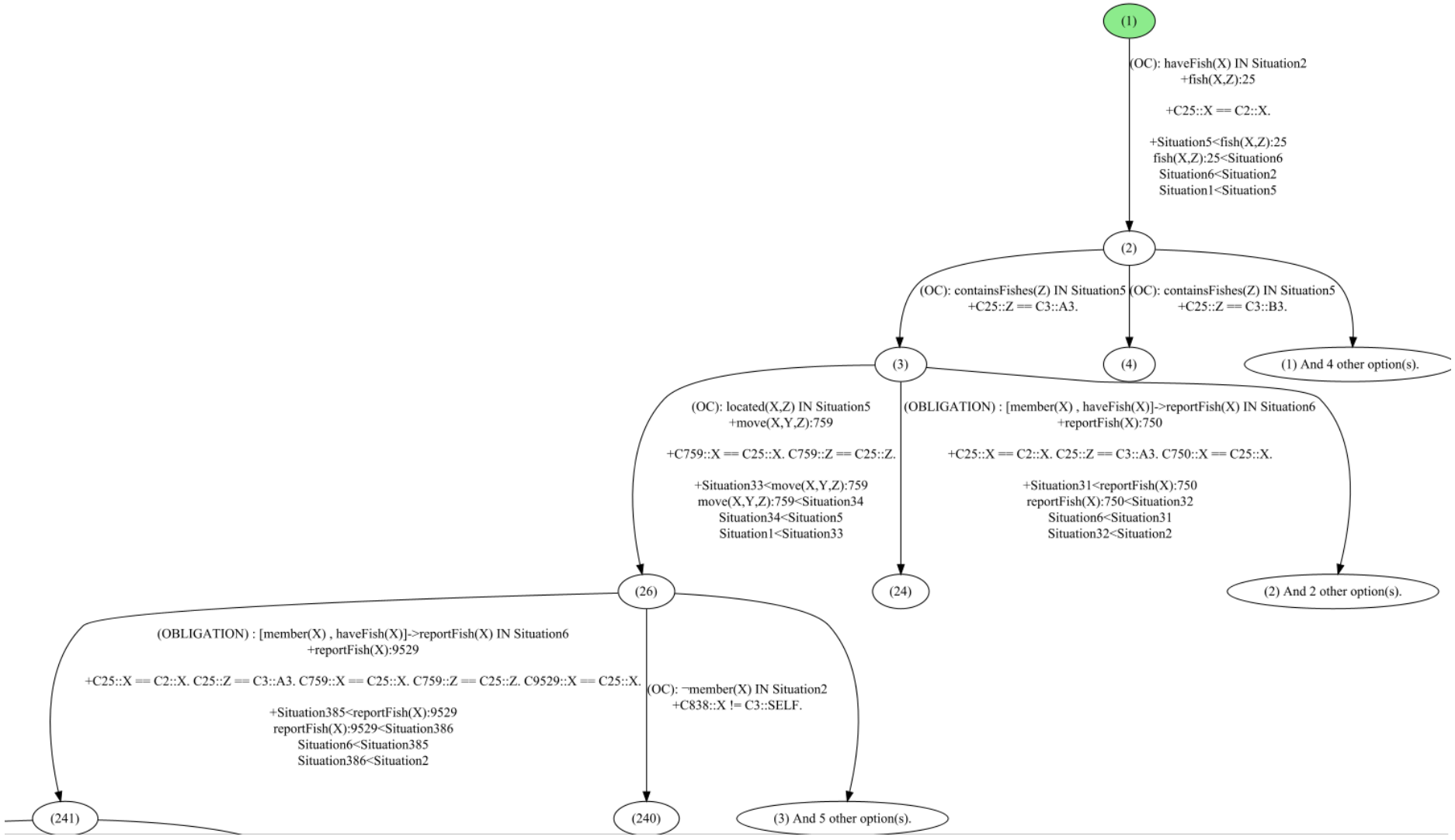
Dans ces annexes seront illustrés les espaces de recherche assez petits pour permettre un aperçu du parcours de l'espace de recherche illustré avec Graphviz. Pour des raisons de lisibilité, les plus grands espaces de recherche ne seront pas affichés.



Annexe 1 : Aperçu d'une portion de l'espace de recherche quand la pêche requiert un filet de pêche, mais que ce filet est seulement permis quand on a une licence.



Annexe 2 : Parcours de l'espace de recherche quand l'objectif est d'avoir du bois, sans normes
(Rouge) : état final, (Jaune): état invalide, (Vert) : état inital.



Annexe 3 : Parcours de l'espace de recherche quand on a l'obligation de déclarer ses prises quand l'agent a fini de pêcher

```
const A1, const A2, const A3, const B1, const B2, const B3, const C1, const C2, const C3, const SELF,
Organization globalOrg{
      Institution : global,
      Assertions : {
            located(SELF, A1),
            areAdjacents(A1, B1), areAdjacents(B1, A1), areAdjacents(B1, C1), areAdjacents(C1, B1),
            areAdjacents(C1, D1), areAdjacents(D1, C1), areAdjacents(A2, B2), areAdjacents(B2, A2),
            areAdjacents(B2, C2), areAdjacents(C2, B2), areAdjacents(C2, D2), areAdjacents(D2, C2),
            areAdjacents(A3, B3), areAdjacents(B3, A3), areAdjacents(B3, C3), areAdjacents(C3, B3),
            areAdjacents(C3, D3), areAdjacents(D3, C3), areAdjacents(A1, A2), areAdjacents(A2, A1),
            areAdjacents(A2, A3), areAdjacents(A3, A2), areAdjacents(A3, A4), areAdjacents(A4, A3),
            areAdjacents(B1, B2), areAdjacents(B2, B1), areAdjacents(B2, B3), areAdjacents(B3, B2),
            areAdjacents(C1, C2), areAdjacents(C2, C1), areAdjacents(C2, C3), areAdjacents(C3, C2),
            areAdjacents(D1, D2), areAdjacents(D2, D1), areAdjacents(D2, D3), areAdjacents(D3, D2)
      },
      Norms : { }
}
```

Annexe 4 : Description de l’organisation global

```
Organization villageOrg{
      Institution : village,
      Assertions : {
         village.member(SELF),.sacred(A1), village.sacred(C1), village.sacred(B3),
         village.protected(B1), village.protected(A3), village.office(B2)
      },
      Norms { }
}
```

Annexe 5 : Description de l'organisation village

```
Organization householdOrg{
      Institution : household,
      Assertions : {
            household.provider(SELF),
            hasTrees(B1), hasTrees(C1),
            hasTrees(C2), hasFish(A3),
            hasFish(B3), hasFish(C3)
      },
      Norms : { }
}
```

Annexe 6 : Description de l'organisation household