UFR Des Sciences Caen — M1 Informatique — 2016/2017
25 Avril 2017

Librairie pour les logiques épistémiques

Projet annuel de la première année de Master

Réalisé par — Ludovic JEAN-BAPTISTE
Tuteur — Bruno ZANUTTINI
Examinateur — Grégory BONNET

Table des matières

Préambule

Lors de la première année de Master Informatique, il était demandé de travailler de façon individuelle sur un sujet libre de notre choix. Il y avait alors deux possibilités : la première étant de proposer un sujet à un enseignant qui pouvait le valider ou non selon son intérêt pour celui-ci, ou bien sinon de choisir son projet parmi une liste de sujets proposés par les enseignants et doctorants de l'université.

Ce sujet de projet a été proposé par Bruno Zanuttini, enseignant à l'université de Caen et chercheur au GREYC (Groupe de recherche en informatique, image, automatique et instrumentation de Caen).

Ce document correspond donc à la synthèse finale de ce projet où il y sera exposé les mécanismes de base de la logique épistémique mais également l'ensemble du travail que j'ai réalisé ainsi ce que j'en ai compris.

1 Introduction

1.1 Qu'est ce que la logique ?

La logique est l'art de formuler des phrases sans ambiguïtés. Elle permet de modéliser et résoudre des problèmes tout comme le raisonnement humain le permet mais avec des vérités et des résultats formels (vrai ou faux / oui ou non). Elle ne peut pas être "mitigée" dans ses résultats puisqu'elle est mécanique et *parfaitement déterministe : ces propositions sont décidables, ce qui signifie qu'elles sont vraies ou fausses en dehors de notre bon vouloir*[1].

C'est de là que la logique s'oppose à l'être humain qui lui a un raisonnement pas toujours très cohérent et une langue ô bien souvent ambiguë : la plupart des mots sont polysémiques et il existe un tas de figure de styles telle que l'ironie qui rendent l'interprétation des phrases plus difficile et surtout pas identique d'un être humain à l'autre. Exemples. Aristote disait "*Tous les hommes sont mortels. Socrate est un homme. Donc Socrate est mortel.*" : dans cet exemple, tout le monde validera sans trop réfléchir ces propos plutôt évidents. Mais que dire de la phrase qui suit : "*Pierre a 300 000 euros sur son livret d'épargne. Donc Pierre est riche*". Des personnes pourraient en effet le penser alors que d'autres non : la réponse étant relative à la situation de chacun mais également elle peut dépendre de ce que l'on connaît de la vie de Pierre : a t-il des dettes à rembourser ? vient-il de vendre sa maison ? Cela nous amène au constat suivant : dans la vie réelle, il existe un nombre infini de propriétés pour décrire notre environnement. La logique, elle, se contentera de travailler sur un environnement beaucoup plus restreint que le logicien, le philosophe ou bien encore l'informaticien aura préalablement défini selon le problème sur lequel il souhaite travailler.

Pour former une théorie logique, il faut tout d'abord fixer un cadre limité, autrement dit un environnement et les propriétés qui le décrivent afin d'enlever toute ambiguïté. Ces propriétés sont ce qu'on appelle des propositions atomiques et peuvent prendre seulement des valeurs booléennes (vrai/faux). Ainsi, par exemple, si notre problème (théorie) chercher à raisonner vis à vis de la météo et de l'humeur de Marie, nous aurions par exemple une proposition *il_fait_soleil* (*Soleil*) et une autre *Marie_est_joyeuse* (*Joyeuse*). Il s'agit des deux seules propriétés fixées par la théorie : l'environnement de la théorie est fixé et plus précisément, nous avons défini l'ensemble $PROP$ des propositions atomiques du langage pour ce problème.

Notre système ne pourra envisager au plus 4 mondes possibles pour $PROP$ ($2^2 = 4$), la figure 1.1 dresse la liste exhaustive de toutes ces combinaisons. Il est également possible de représenter le monde réel (ou monde courant) en donnant à chacune des propositions une valeur (également nommée valuation). J'en reparlerais davantage dans la section traitant les modèles pour la logique épistémique.

A partir de ces propositions, le langage contient des opérateurs afin de formuler des phrases, par exemple les formules ψ, ϕ et ω :

[1] Propos tirés du livre *Faussement vrai et vraiment faux: la logique au quotidien* par Solange Cuénod (p. 232).

Soleil	*Joyeuse*
vrai	vrai
vrai	faux
faux	vrai
faux	faux

Table 1: Mondes possibles : toutes les combinaisons possibles pour $PROP$

- $\psi \equiv Soleil \vee Joyeuse$
  Il fait soleil ou Marie est joyeuse.
- $\phi \equiv Soleil \Rightarrow Joyeuse$
  Si il fait Soleil alors Marie est Joyeuse.
- $\omega \equiv \neg Joyeuse$
  Marie n'est pas joyeuse.

Au passage, on peut remarquer que la conjonction des trois formules précédentes $\psi \wedge \phi \wedge \omega$ est toujours fausse : c'est une *antologie*. A l'inverse, on parle de *tautologie* d'une formule qui est toujours vraie quelque soit les valeurs des valuations de ses propositions (on dit aussi qu'elle est *valide*). Aussi, on dit qu'une formule est *satisfaisable* si il existe au moins un ensemble de valeurs données aux propositions qui rendent la formule vraie.

Ajoutons que la logique aujourd'hui permet de modéliser et formaliser beaucoup de problèmes complexes afin de les résoudre en lieu et place des algorithmes spécifiques. C'est le cas du jeu du *SUDOKU* où il est possible de modéliser les contraintes que le jeu impose avec des propositions (clauses SAT) que l'on donne ensuite à un solver (solver SAT). Ce projet est aussi une preuve que la logique a d'autres utilités.

Les propositions atomiques, leurs valuations ainsi que les opérateurs pour composer des formules logiques peuvent être accompagnés de modalités et c'est ainsi que nous en venons à se demander qu'est-ce qu'une logique modale ?

1.2 Une logique modale ?

La logique dite classique correspond au premier formalisme de la logique qui a été créé au début du XX^e siècle. D'autres sont arrivées après, c'est pourquoi on parle de logique classique.

La logique classique comprend entre autres les catégories de logiques suivantes :

Logique propositionnelle C'est la logique qui est évoquée dans la section 1.1. Elle se base uniquement sur des propositions atomiques booléennes. On parle de logique d'ordre 0.

Logique des prédicats Il s'agit d'une logique plus expressive mais beaucoup plus complexe d'un point de vue calculatoire. On parle de logique de premier ordre.

La logique modale est une extension de la logique propositionnelle classique où l'on raisonne sur l'incertain et envers des situations évolutives (logique temporelle, logique de la connaissance, de la croyance, logique déontique, etc. . .).

Elle permet d'aller plus loin dans le raisonnement et de s'ouvrir à des problèmes non factuels comme devaient être les problèmes de la logique propositionnelle.

Une logique modale s'appuie — comme son nom l'indique — sur des modalités qui s'ajoutent au langage initial de la logique propositionnelle. Dans ce projet, la logique épistémique est donc une logique modale qui possède deux modalités phares : celle de la connaissance et celle de la connaissance commune d'agents. Nous entrerons dans les détails dans la partie suivante.

1.3 Sujet et déroulement du projet

Maintenant que le cadre est posé, il est venu le moment d'expliquer ce qu'est le sujet du projet que j'expose dans ce rapport et comment s'est-il déroulé.

Sujet L'objectif était de réaliser une librairie en Java *permettant de manipuler les formules et leur sémantique, et fournissant des algorithmes de raisonnement*[2] à partir de modèles épistémiques et d'actions (appelé aussi évènement).

Déroulement La première étape était de comprendre les différents principes de raisonnement à partir de cette logique notamment les modalités de connaissance dans les formules, les modèles de Kripke (que l'on évoquera dans la suite de ce rapport) ainsi que les règles de *model checking* existantes.

Une fois les différentes notions acquises, venait l'heure de réaliser la conception objet de la librairie afin de représenter . . .

- les formules logiques
- les modèles de Kripke

. . . puis de coder tout cela en langage Java.

La tâche suivante était d'implémenter les mécanismes de *model checking* et l'algorithme de produit de mise à jour entre un modèle épistémique et un modèle d'évènement (que l'on verra également par la suite). Ce dernier étant relativement lourd, la première chose était de le faire fonctionner via une implémentation naïve puis de le rendre plus efficace d'un point de vue algorithmique.

Enfin, il était nécessaire de modéliser des expériences épistémiques connues avec la librairie nouvellement créée et de voir si les résultats obtenus sont bien en accord avec ceux qui sont attendus théoriquement. Cette étape fera office de tests pour valider (ou non) le travail effectué.

[2] Ces quelques mots en italique proviennent directement de l'énoncé du sujet.

Selon le temps restant jusqu'au jour de la soutenance, l'idée serait d'aller plus loin en ce qui concerne la logique épistémique dynamique.

2 La logique épistémique

La logique épistémique — qu'on appelle aussi logique de la connaissance — est une logique modale comme il en existes d'autres (déontique, temporelle...). Elle est le sujet de travaux de recherche en informatique notamment en Intelligence Artificielle (mais pas seulement). Elle figure — entre autres — dans le cadre des systèmes multi-agents et la communication entre agents : elle permet de modéliser la connaissance d'agents au vu de leurs observations et des évènements qui se produisent sur leur environnement. D'un point de vue communication, la logique épistémique ne s'intéresse pas de savoir où vient l'information mais cherche à travailler avec le contenu du message. On parle d'annonces dès lors qu'une information est donnée à un ou plusieurs agents (annonce publique quand elle est donnée à tous les agents), et auront comme conséquence de modifier l'état épistémique du problème. Les mécanismes implémentés dans ce projet seront expliqués dans la partie suivante.

Un scénario typique qui permet de bien comprendre ce qu'est le raisonnement sur la connaissance est le suivant :
Vous êtes au volant de votre voiture quand vous arrivez à une intersection sans signalisation particulière. A votre gauche, une voiture qui s'arrête vous laissant passer en premier. La priorité à droite s'applique dans cet exemple et les usagers respectent le code de la route. Vous passez donc l'intersection en premier. Que se passe-t-il d'un point de la connaissance pour cette situation ?

Lorsque vous arrivez à l'intersection, vous savez que vous êtes prioritaire. Vous savez également que l'automobiliste à gauche sait que vous avez la priorité et que vous êtes sûr qu'il ne passera pas à votre place car vous savez qu'il sait que vous savez qu'il sait que vous êtes prioritaire. Cela peux ne jamais en finir... On parle alors de connaissance commune quand un fait est connu pour chacun des agents (ici les 2 automobilistes). Dans cette expérience, il n'y a aucun doute pour les agents (vous et l'autre usager de la route) que vous avez la priorité. La connaissance commune permet dans certaines situations de décider d'une action qui peut sembler être banale pour l'homme mais dans le cadre d'un système multi-agent qui doit être programmé (c'est à dire des agents qui ont un fonctionnement totalement indépendant des autres), elle permet de prendre des décisions et c'est là tout l'enjeu de la logique épistémique.

L'un des objectifs de la logique épistémique est de traiter les connaissances d'un seul agent, celles partagées entre certains agents et enfin les situations de connaissance commune. Pour parvenir à raisonner au travers de ces concepts, la logique épistémique a un langage propre qui se base sur celui de la logique propositionnelle avec les modalités en plus. Mais également, ces mécanismes s'appuient sur des modèles qui représentent l'état d'une situation épistémique et ce que les agents peuvent confondre (manque de connaissance).

2.1 Langage

Le langage de la logique épistémique, noté $\mathcal{L}_{el}$ est composé d'abord d'un ensemble d'agents AGT et d'un ensemble de propositions atomiques $PROP$.

Formules Les règles élémentaires de construction des formules dans le langage $\mathcal{L}_{el}$ sont :

- pour toute proposition atomique $p \in PROP$, p est une formule ;
- $\top$ (*vrai*) est une formule ;
- si φ est une formule alors $\neg\varphi$ est une formule ;
- si φ est une formule et ψ est une formule alors la *disjonction* $(\varphi \vee \psi)$ est une formule ;
- si φ est une formule alors pour tout agent $a \in AGT$, $K_a\varphi$ qui signifie "*l'agent a sait que* φ" est une formule.

Déductions De ces cinq règles élémentaires (axiomes), il est possible de définir d'autres constructions pour $\mathcal{L}_{el}$ à partir des règles de déduction définies par la logique classique.

- $\bot$ (*vrai*) est déduit de $\neg\top$;
- la *conjonction* $(\varphi \wedge \psi)$ est déduit de $\neg(\neg\varphi \vee \neg\psi)$;
- $(\varphi \implies \psi)$ qui signifie "φ *implique* ψ" est déduit de $(\neg\varphi \vee \psi)$;
- $(\varphi \iff \psi)$ qui signifie "φ *est équivalent à* ψ" est déduit de $((\varphi \implies \psi) \wedge (\psi \implies \varphi))$;
- $\widehat{K}_a\varphi$ qui signifie "*l'agent a envisage* φ" est déduit de $\neg K_a \neg\varphi$;
- Pour un ensemble d'agents $E \subseteq AGT$,
  $CK_E\ \varphi$ qui signifie "*la connaissance commune*[3] *de* φ *pour les agents de l'ensemble* E".

Modalités On peut le deviner au regard de nos axiomes et de nos règles déduites, il existe pour la logique épistémique — qui rappelons le est une logique modale — deux principales modalités pour raisonner sur la connaissance, à savoir :

- La modalité pour la connaissance d'un agent a : $K_a\varphi$;
- La modalité pour la connaissance commune d'un ensemble d'agents E : $CK_E\ \varphi$.

Exemples Reprenons l'exemple donné en introduction avec les propositions *Soleil* (*il fait soleil*) et *Joyeuse* (*Marie est joyeuse*). Grâce au langage $\mathcal{L}_{el}$ doté d'un ensemble d'agents $AGT = \{Antoine, Marie\}$, et d'un ensemble de propositions $PROP = \{Soleil, Joyeuse\}$, il est possible d'écrire :

- $K_{Marie}\ Joyeuse$
  Marie sait qu'elle est joyeuse.
- $\widehat{K}_{Antoine}\ \neg Joyeuse \wedge \neg Soleil$
  Antoine envisage que Marie n'est pas joyeuse et qu'il ne fait pas soleil.

[3] CK pour Common Knowledge en anglais

2.2 Mondes possibles et modèles de Kripke

Mondes possibles En introduction, nous avons évoqué l'existence de mondes possibles. Pour les deux propositions (*Soleil* et *Joyeuse*) que nous avions défini, il y avait 4 mondes possibles. Ces mondes peuvent être alors confondus par les agents selon leurs connaissances. La sémantique des mondes possibles est issue du travail de trois logiciens : *Saul Kripke*, *Stig Kanger* et *Jaakko Hintikka* dans les années 1950. L'idée qui repose derrière cette sémantique est qu'au delà de l'état réel (monde réel) d'une situation S, on peut distinguer d'autres mondes considérés comme possibles au vu des propriétés de S en jouant avec leurs valeurs de vérité.

Considérons toujours notre agent *Marie* et les deux propositions *Soleil* et *Joyeuse*. Imaginons la situations suivante :
Marie se trouve enfermée dans une pièce sans lumière ni fenêtre. Elle sait bien que cette situation ne la rend pas joyeuse ($\neg Joyeuse$) et elle n'a en conséquence aucune information sur la météo. Elle peut imaginer que le soleil brille ($Soleil$) tout comme elle peut en douter ($\neg Soleil$). Ce doute, plus simplement ce manque de connaissance met en exergue deux mondes possibles qui sont reliés par une relation d'accessibilité pour cet agent, *Marie* (Figure 1). Lorsque l'on se place dans un monde, l'agent peut avoir des doutes à partir de ce monde mais également considérer ce monde même comme possible : c'est le cas dans la figure 1 qui contient des relations réflexives.

Modèle de Kripke Étant donné nos ensembles AGT (agents) et $PROP$ (propositions) déjà définis préalablement, un modèle de Kripke $\mathcal{M}$ est donc un triplet (W, R, V) où :

- W est un ensemble de monde (états) possibles, dit *univers* ;
- $R : AGT \to 2^{W \times W}$ qui associe à chaque agent des relations d'accessibilité entre les mondes ;
- $V : W \to 2^{PROP}$ est une fonction de valuation qui associe à chaque monde l'ensemble des valeurs de vérité (vrai/faux) pour chaque proposition.

Modèle pointé On appelle modèle pointé un couple $(\mathcal{M}, w)$ où $\mathcal{M}$ est un modèle de Kripke (W, R, V) et $w \in W$ un monde de ce modèle.

2.3 Évènements complexes

Les évènements complexes au travers des modèles d'évènements nous emmènent dans ce qu'on appelle la **logique épistémique dynamique**. Dynamique car les modèles épistémiques (de Kripke) seront amenés à évoluer selon les évènements qui leurs seront appliqués.

Annonces Les annonces correspondent aux informations que les agents apprennent au fur et à mesure de l'évolution d'une situation épistémique. Elles peuvent être publiques et dans ce cas son modèle associé est seulement composé d'un état auquel on associe une vérité (une formule logique). La Figure 2 est le modèle d'une annonce quelconque φ pour un problème avec 3 agents a, b et c (exemple).

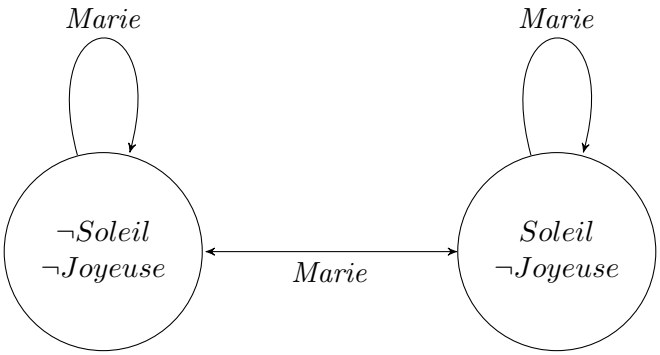


Figure 1: Monde possibles pour l'agent *Marie*

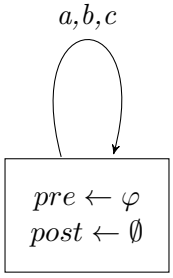


Figure 2: Modèle de l'annonce publique de φ pour les agents a, b et c

Modèles d'évènements Les modèles d'évènements sont des modèles qui se rapprochent inexorablement des modèles de Kripke par leur structure. En effet, il s'agit aussi d'un graphe avec des états et des relations d'accessibilité pour les agents. La différence se trouve dans le contenu même des mondes : il n'y aura plus un ensemble de valuations pour les propositions comme c'était le cas pour les mondes épistémiques mais plutôt :

- Une **pré-condition** : il s'agit d'une formule φ qui activera l'action pour un modèle pointé $(\mathcal{M}, w)$ si $\mathcal{M}, w \models \varphi$. Nous verrons dans le point suivant quelles sont les règles qui permettent de décider si une formule est satisfaite pour un modèle pointé (règles de satisfaisabilité).
- Une **post-condition** (optionnelle) : c'est les modifications apportées à la valuation de w après l'activation de l'action (pour 0, 1 ou plusieurs propositions de $PROP$).

Plus simplement, une action est décrite par deux éléments : une condition d'activation (pré-condition) et un ensemble de modifications à apporter en cas d'activation (post-condition) .

Remarque Il n'y a pas de post-condition dans le cas d'une simple annonce publique (Figure 2).

2.4 Règles de satisfaisabilité (*model checking*)

Avec les pré-conditions que contiennent nos mondes d'évènements, nous devons être en mesure d'en évaluer leur valeur de vérité à partir d'un modèle pointé donné. Pour y parvenir, des règles de satisfaisabilité existent dans la littérature et le but sera de les implémenter dans la librairie, point essentiel du projet.

Règles La vérification des modèles[4] s'effectue en suivant des règles. En considérant encore une fois $PROP$, AGT, un modèle pointé $(\mathcal{M}, w)$ et en rappelant qu'un modèle épistémique de Kripke est un tuple (W, R, V), nous avons les règles suivantes :

- $\mathcal{M}, w \models p$ ssi $p \in V(w)$;
- $\mathcal{M}, w \models \neg\varphi$ ssi $\mathcal{M}, w \not\models \varphi$;
- $\mathcal{M}, w \models (\varphi \wedge \psi)$ ssi $\mathcal{M}, w \models \varphi$ et $\mathcal{M}, w \models \psi$;
- $\mathcal{M}, w \models (\varphi \vee \psi)$ ssi $\mathcal{M}, w \models \varphi$ ou $\mathcal{M}, w \models \psi$;
- Pour un agent $a \in AGT$,
  $\mathcal{M}, w \models K_a\varphi$ ssi pour tout $w' \in R_a(w)$, on a $\mathcal{M}, w' \models \varphi$;
- Pour un ensemble $E \subseteq AGT$,
  $\mathcal{M}, w \models CK_E\ \varphi$ ssi pour tout agent $a \in E$, pour tout $w' \in R_a(w)$, on a $\mathcal{M}, w' \models \varphi$;

Remarque L'ensemble des règles énoncées ci-dessus (hormis la première qui correspond au point d'arrêt) est déclenchable par induction. On réutilise sans cesse les mêmes règles tant que l'on n'a pas parcouru l'intégralité de la formule φ (dans le pire cas).

Validité Notons qu'une formule φ est dite valide dans un modèle $\mathcal{M}(W, R, V)$ si pour tout $w \in W$, on a $\mathcal{M}, w \models \varphi$. On dit simplement qu'elle est valide si elle est valide pour tout modèle.

3 Réalisation d'une librairie

3.1 Conception

Nous avons pratiquement tout ce qu'il faut pour réaliser une petite librairie en Java. La conception est une phase importante. Dans notre cas, elle reste tout de même assez simple car il n'y a pas tant de paramètres à prendre en compte.

Le code source du projet Java se décompose en deux principaux paquetages : le premier met à disposition un ensemble de classes pour construire nos formules. Le second, lui, propose les différents composants pour représenter nos modèles de Kripke.

[4] *model checking* en anglais et le système qui effectue ces vérifications est appelé *model checker*

3.1.1 Fabrication des formules

La logique épistémique est régit par un langage, noté $\mathcal{L}_{el}$ dont le détail de construction des formules est donné dans la section 2.1. L'objectif de la librairie est de proposer une structure de données en accord avec ce langage. Comme nous l'avons vu, la grammaire est définie par induction à partir d'une proposition notée p (axiome de base).

Ainsi, nous aurons autant de classes que d'opérateurs de ce langage. La négation, la conjonction, la disjonction mais également les modalités de connaissances ($K_a\varphi$ et $CK_E\varphi$) seront chacun représentés par une classe.

Puisqu'une formule se construit à la manière d'un arbre où les opérateurs et les propositions correspondent respectivement à ses nœuds et ses feuilles, le *design pattern* `Composite` est la solution. Chaque classe du package `formule` hérite de la classe abstraite `Formule` et redéfinit ses méthodes. Les constructeurs de ces classes acceptent de zéro à plusieurs sous-membres (sous-formules). Ci-dessous, les entrées possibles pour nos composants de formules :

- Les propositions et les valeurs constantes Vrai (`Vrai`) et Faux (`Faux`) ne prennent **aucune** sous-formule.
- La négation (`Non`), les modalités épistémiques (`SaitQue`, `Envisage` et `CC`) prennent **une** sous-formule.
- L'implication ($\Longrightarrow$) prend **deux** sous-formules.
- La disjonction ($\vee$) et la conjonction ($\wedge$) prennent **au moins deux** sous-formules.

Remarque Dans la section 2.1, la conjonction et la disjonction sont présentées comme des relations binaires (deux sous-formules). Mais si nous étions amenés à modéliser une formule conjonctive avec plusieurs clauses, cela pourrait devenir pénible d'écrire `new Et(a, new Et(b, new Et(c, new Et(d, e))))`, en considérant `a, b, c, d, e` comme des formules. Fort de ce constat lors de la modélisation des problèmes épistémiques qui seront présentés à la fin de ce rapport, j'ai ajouté en dernier lieu à cette librairie la possibilité d'écrire directement `new Et(a, b, c, d, e)` puisqu'ils s'agit de relations transitives. Le but d'une librairie est avant tout de proposer des outils utiles, fonctionnels mais également pratiques d'utilisation !

Diagramme UML L'annexe A de ce document présente le diagramme UML du package `formule`.

3.1.2 Représentation des modèles

L' autre package de notre librairie s'intéresse à la représentation de nos modèles épistémiques et d'évènements. Elle s'articule autour d'une classe principale générique `Modele<T extends Monde>` qui permet la modélisation d'un graphe orienté avec des sommets (mondes) et des arêtes (relations d'accessibilité). La distinction entre un modèle épistémique et un modèle d'évènement se trouve sur le choix du type de sommets de ce graphe.

Epistemique vs Evenement En effet, soit nous souhaitons créer le modèle d'une situation épistémique et nous avons alors besoin d'utiliser la classe `MondeEpistemique`, soit nous voulons construire un évènement complexe et alors nous utiliserons la classe `MondeEvenement`. La différence entre ces deux classes se trouve dans leur contenu : d'un côté un ensemble de valuations et de l'autre une pré-condition et post-condition. Bien évidemment, ces deux classes héritent d'une classe vide `Monde`, type exigé par la classe template `Modele`.

Liste de successeurs Dans un premier temps, les relations étaient stockées simplement dans une liste d'objets `Relation(agent, mondeA, mondeB)`. Dès lors que les problèmes devenaient plus grands, le temps d'exécution n'étaient plus acceptables notamment pour l'algorithme du produit de mise à jour qui sera détaillé dans la prochaine section. Ainsi, pas d'autres choix que de mieux structurer nos relations en mémoire et pour cela il existe les listes de successeurs où pour chaque agent et chaque monde, on établit une liste des mondes que l'on peut atteindre (successeurs). Si l'on veut vérifier l'existence d'une relation entre un monde d'origine mA, un monde d'arrivée mB et cela pour un agent donné i, il suffit de se positionner dans le dictionnaire de l'agent i et de parcourir la liste en clé mA. Trouver une clé dans un dictionnaire se faisant en temps constant (merci les tables de hashages), il reste seulement le parcours de la liste de successeurs qui s'effectue en moyenne en $\mathcal{O}(n/2)$ et au pire en $\mathcal{O}(n)$ (où n est le nombre de successeurs pour un monde et un agent donné).

Remarque La matrice d'adjacence est une autre solution de représentation en mémoire des relations qui permet une recherche en temps constant mais elle requiert un espace en mémoire bien plus important : $\mathcal{O}(|W^2|))$ où W est l'ensemble des mondes. Elle reste tout de même une représentation sérieuse lorsque le nombre d'arêtes est plus important que le nombres de sommets (éviter les matrices creuses). Dans notre cas, ce n'est pas forcément vrai d'où le choix d'utiliser les listes de successeurs.

3.2 Algorithmes

Le développement d'une librairie n'est pas seulement de la conception : il s'agit aussi de proposer des outils répondant à des tâches diverses : l'implémentation de mécanismes pour la résolution de problèmes épistémiques est partie intégrante du projet.

3.2.1 Model checking

La section 2.4 présentait les différentes règles pour vérifier si une pré-condition est satisfaite pour un modèle pointé donné. D'un point de vue du code, et au vu de la construction de nos formules, une méthode abstraite de la classe mère `Formule` répondra à ce problème de décision : il s'agit de la méthode publique `boolean satisfaite(Modele<MondeEpistemique> modele, MondeEpistemique mondeRef)` qui est redéfinie dans chaque composant de formule. Son premier argument est le modèle de Kripke sur lequel on souhaite faire le test et le second est un monde de ce modèle : cela correspond à la définition d'un modèle pointé. A noter que cette méthode lève une exception `ModeleNonPointeException` si le monde spécifié n'appartient pas au modèle donné.

Récursivité Comme déjà expliqué, une formule est construite par induction à partir des règles de base de la logique. Le problème de décision qui prend une formule et regarde si elle est satisfaite pour un modèle pointé est un problème qui peut être implémenté avec la récursivité. En effet, prenons le cas de l'évaluation de la connaissance (ou non) pour un modèle pointé $\mathcal{M}, w$, au travers de la modalité $K_a\varphi$ qui se construit avec la librairie par l'instruction suivante : `new SaitQue(a, φ)`. Cette formule est satisfaite pour $\mathcal{M}, w$ et pour l'agent a si φ est satisfaite dans l'ensemble des mondes successeurs de w (notés $w_1, w_2, ..., w_n$) pour l'agent a. Cela passe donc par autant d'appels de la méthode `satisfaite` qu'il y a de successeurs (c'est à dire n) mais cette fois pour la formule φ (la modalité a été consommée). Si φ est une formule composée, d'autres appels auront lieu. Si il s'agit d'une proposition, on regardera si elle est vraie dans l'ensemble des valuations du monde courant $(w_1, w_2, ..., w_n)$ et le résultat sera remonté (condition d'arrêt). C'est le principe même de la récursivité au détail près que la méthode `satisfaite` appelée n'est pas la même selon le type de formule à évaluer.

3.2.2 Produit de mise à jour

Tout au long de ce rapport, il a été évoqué à plusieurs reprises les modèles de Kripke qui décrivent une situation épistémique et les modèles d'évènements qui peuvent aussi bien modéliser une simple annonce publique tout comme un évènement plus complexe. Mais toujours rien vis à vis de la résolution. Que fait-on de ces modèles pour arriver à notre fin, celle d'obtenir une solution à un problème épistémique ?

Cela passe par ce qu'on appelle le **produit de mise à jour** d'un couple composé d'un modèle épistémique $\mathcal{M}$ et d'un modèle d'évènement $\mathcal{E}$. Le résultat de ce produit est un nouveau modèle épistémique $\mathcal{M}'$ qui correspond au modèle $\mathcal{M}$ auquel on a appliqué l'évènement $\mathcal{E}$.

Règles d'application Le produit de mise à jour $\mathcal{M} \otimes \mathcal{E} = \mathcal{M}'$ où $\mathcal{M}$, $\mathcal{E}$ et $\mathcal{M}'$ sont définis par leurs composants respectifs $(W^{\mathcal{M}}, R^{\mathcal{M}}, V^{\mathcal{M}})$, $(W^{\mathcal{E}}, R^{\mathcal{E}}, Pre^{\mathcal{E}}, Post^{\mathcal{E}})$, et (W', R', V') est régit par les règles ci-dessous.

- $(w' \leftarrow (w, e)) \in W'$ si $w \in W^{\mathcal{M}}$, $e \in W^{\mathcal{E}}$ et $\mathcal{M}, w \models Pre_e^{\mathcal{E}}$;
  Le monde w' qui correspond à l'association de w et e appartient à W' si w et e appartiennent respectivement à $W^{\mathcal{M}}$ et $W^{\mathcal{E}}$ et que la pré-condition de e est satisfaite dans le modèle pointé $\mathcal{M}, w$.

- $w'_1 \leftarrow (w_1, e_1)$, $w'_2 \leftarrow (w_2, e_2)$ tel que $(w'_1, w'_2) \in R'_a$ si $w, w' \in W^{\mathcal{M}}$, $e, e' \in W^{\mathcal{E}}$, $(w_1, w_2) \in R_a^{\mathcal{M}}$ et $(e_1, e_2) \in R_a^{\mathcal{E}}$ pour tout a ;
  La relation entre mondes résultants (w'_1, w'_2) existe dans R'_a si il existe la relation (w_1, w_2) dans $R_a^{\mathcal{M}}$ et il existe la relation (e_1, e_2) dans $R_a^{\mathcal{E}}$ pour tout agent a.

- $Post(e) \subseteq V'_{w'}$ pour tout $(w' \leftarrow (w, e)) \in W'$;
  La post-condition e a été appliquée à tout monde w' appartenant à W' qui résulte d'un couple (w, e) en conséquence à la première règle.

Implémentation Retour au code, il fallait donc implémenter cet algorithme en Java. Il se trouve dans la classe `Evenement` du package `modeles` au travers d'une méthode statique `produitMAJ(Modele<MondeEpistemique>, Modele<MondeEvenement>)` qui retourne un modèle épistémique : il ne met pas à jour le modèle initial mais renvoie un nouveau modèle tout juste créé en mémoire.

Par ailleurs, deux méthodes statiques dans cette classe accompagnent le produit de mise à jour. Il s'agit de `annoncePublique(Modele<MondeEpistemique>, Formule formule)` qui permet de créer le modèle d'une annonce publique en reprenant la formule spécifiée en argument et de `annoncePrivee(Modele<MondeEpistemique>, Agent agent, Formule formule)` qui permet de créer une annonce à tous les agents sauf un. J'ai créé cette classe avec l'idée d'ajouter quelques modèles d'évènements pré-établis mais il ne sont que deux au moment de l'écriture de ce rapport. Bien entendu, il est possible de créer des modèles d'évènement spécifiques sans passer par ces méthodes !

Algorithm 1 Produit de mise à jour

Entree : $\mathcal{M}$: modèle épistémique initial
Entree : $\mathcal{E}$: modèle d'évènement
Sortie : $\mathcal{M}'$: modèle épistémique résultant

```
couples ← nouveau Dictionnaire()
pour tout w ∈ mondes(M) faire
  pour tout e ∈ mondes(E) faire
    si Pre(e) est satisfaite par (M, w) alors
      w' ← nouveau MondeEpistemique()
      ajouterValuations(w', valuations(w))
      mettreAJour(w', Post(e))
      ajouterMonde(M', w')
      couples[(w, e)] ← w'
    fin si
  fin pour
fin pour
pour tout a ∈ agents(M) faire
  pour tout (w, e) ∈ couples faire
    w' ← couples[(w, e)]
    pour tout i ∈ successeurs(w, a, M) faire
      pour tout j ∈ successeurs(e, a, E) faire
        w'' ← couples[(i, j)]
        si w'' ≠ ∅ alors
          ajouterRelation(M', a, w', w'')
        fin si
      fin pour
    fin pour
  fin pour
fin pour
retourner M'
```

Algorithmique Une des attentions que l'on porte lorsque l'on élabore un algorithme est sa complexité. Comme déjà dit dans la section 3.1.2, il y avait de très longs temps d'exécutions lors de la première version réalisée car l'idée était avant tout de créer une version naïve qui fonctionnait. La seconde version, cette fois avec les listes de successeurs, est bien plus efficace et cela se ressent dans les problèmes que j'ai modélisé (comme par exemple *The Sum-and-Product Riddle* qui sera présenté dans la dernière partie). L'algorithme 1 présente l'algorithme final en pseudo-code.

4 Modélisation de problèmes

Tout au long du projet, les algorithmes présentés au dessus ainsi que tout le code Java a été testé à partir de deux problèmes épistémiques connus à savoir *The Muddy Children Puzzle* et *The Sum-and-Product Riddle*. Dans cette dernière partie, il sera question de présenter ces problèmes ainsi que de présenter les modèles de Kripke associés à chaque étape de la résolution (annonces).

4.1 *The Muddy Children Puzzle*

Le problème des enfants sales (*muddy children*) est le suivant :

> Des enfants jouent dans la boue. A leur retour, leur père dit à ses n enfants la phrase suivante : "*au moins l'un d'entre vous à le front sale de boue*". Bien entendu, chacun des enfants ne peut pas voir son propre front mais voit celui des autres. Parmi les n enfants, k sont sales (dans cet exemple, on prendra seulement $n = k = 2$ mais dans le cas général, on a $n \geq k \geq 1$.
>
> Première étape. Le père demande : "*L'un d'entre vous connait-il l'état de son front ? Si oui, levez la main*". Personne ne lève la main.
>
> Deuxième étape. Le père pose de nouveau la question : "*L'un d'entre vous connait-il l'état de son front ? Si oui, levez la main*". Cette fois, chacun des enfants lève la main. Ils sont tous les deux sales.

Modèle initial Que s'est-il passé entre la première et deuxième étape ? Décrivons notre problème épistémique. Dans cet exemple, tout d'abord, nous avons $n = k = 2$ agents (A et B). Nous avons donc 2 propositions notées m_a et m_b et qui signifient "*l'enfant A est sale*" et "*l'enfant B est sale*". Ainsi, avec la sémantique des mondes possibles et de Kripke, on peut en déduire le modèle épistémique initial (Figure **??**). A ce moment, les agents voient seulement le front de l'autre et ne savent rien !

Après la première annonce $m_a \vee m_b$ Les agents savent désormais que l'un d'entre eux est sale (ou les deux). Parmi les 4 mondes possibles initiaux, un ne permet pas à la formule $m_a \vee m_b$ d'être satisfaite. Ce monde $(\neg m_a, \neg m_b)$ est donc retiré puisqu'il n'est pas compatible avec la pré-condition. La figure 4 montre le nouveau modèle suite à cette annonce.

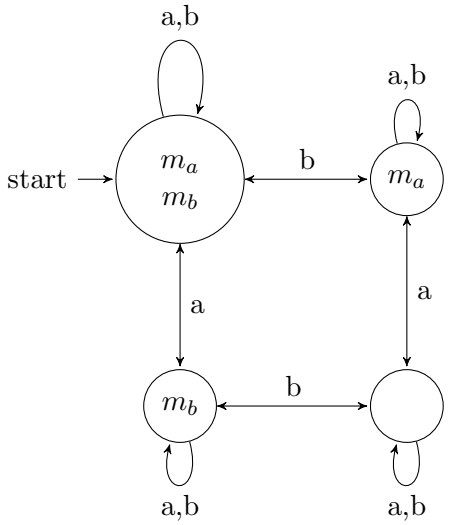


Figure 3: Modèle initial du *Muddy Children Puzzle* avec 2 enfants

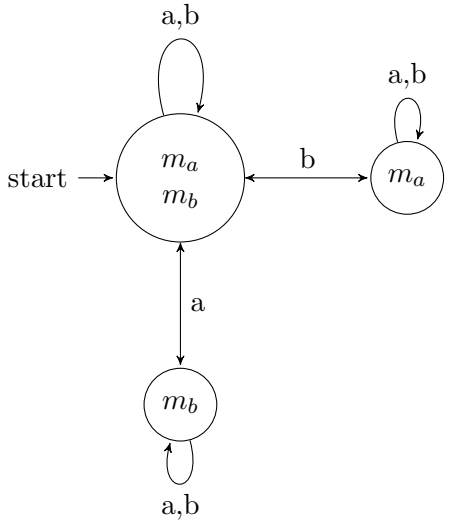


Figure 4: Modèle mis à jour après l'annonce $m_a \vee m_b$

Après la seconde annonce $\neg K_a m_a \wedge \neg K_b m_b$ Lorsque le père demande aux enfants de lever la main la première fois, aucun ne lève sa main. Cela correspond à une annonce pour chacun des agents, à savoir celle où on apprend que ni A, ni B ne connait l'état de son propre front. Or, dans la figure 4, on peut remarquer que dans le monde $(m_a, \neg m_b)$ en haut à droite, l'agent a imagine seulement ce monde : cela voudrait dire qu'il sait l'état de son front. C'est incohérent avec la formule tout juste annoncée : la pré-condition n'est pas satisfaite et le monde est ignoré dans le produit de mise à jour. On retrouve le même raisonnement dans le monde $(\neg m_a, m_b)$ en bas à gauche pour l'agent b. On se retrouve seulement avec le monde (m_a, m_b). La Figure 5 présente le dernier monde de ce problème : c'est la solution ! Les deux enfants sont bien sales et c'est pourquoi ils lèvent tous les deux la main. Le modèle de Kripke est bien en accord avec leur état de connaissance.

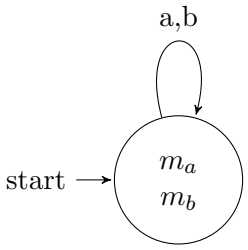


Figure 5: Modèle mis à jour après l'annonce $\neg K_a m_a \wedge \neg K_b m_b$

Cas général Dans le cas général, il faudra autant d'annonces que d'enfants pour atteindre la solution : à chaque étape, un agent de plus apprendra son état et lèvera la main.

4.2 *The Sum-and-Product Riddle*

Cette énigme a été énoncé dans le journal de mathématiques néerlandais *Nieuw Archief voor Wiskunde* en 1969 et il a été résolu l'année suivante. L'énonce est le suivant :

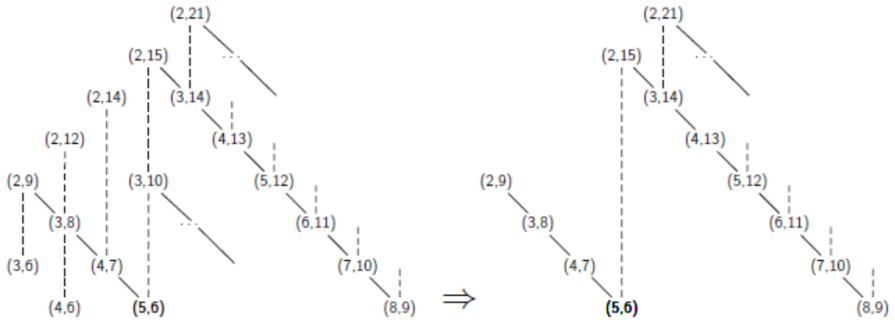


Figure 7: Modèle (partiel) mis à jour après l'annonce de $K_S(\bigwedge_{x,y} \neg K_P\ x \vee y)$

Un père et ses deux fils S (pour somme) et P (pour produit) se trouvent ensemble. Le père dit : *j'ai choisi deux nombres* x,y *tels que* $1 < x < y$ *et* $x + y \leq 100$. Je vais dire à S la valeur de la somme $x + y$ et je vais informer P du produit xy.

Un échange entre les deux agents à lieu :
P : *Je ne connais pas* x *et* y.
S : *Je le savais.*
P : *Je connais* x *et* y.
S : *Moi aussi.*
La solution prouvera qu'ils sont en mesure de connaître les deux nombres à l'issue de cet échange !

Modèle initial Construction du modèle initial : il s'agit de tous les mondes formant tous les couples solutions possibles x, y. Une façon de faire est de définir une proposition par nombre et de rendre *vrai* exactement deux propositions par monde (valuations). La Figure[5] 6 montre le modèle partiel de ce problème où la somme vaut 11 : les relations (transitives) de l'agent P sont en pointillés et S en ligne pleine. Les relations réflexives sont omises dans cette figure pour que ce soit plus lisible.

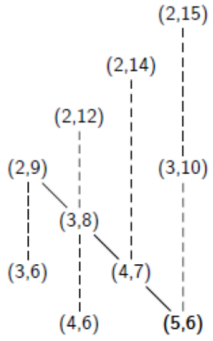


Figure 6: Modèle (initial) où les sommes valent 11

Après la première annonce $K_S(\bigwedge_{x,y} \neg K_P\ x \vee y)$ La première annonce est double : d'une part elle informe que P ne connait pas les deux nombres mais également que S le sait. Après cette annonce, on passe de 2352 mondes possibles à 145 et de 83054 relations (tout agents confondus) à 2714. Même si un temps de calcul existe (3-4 secondes) pour effectuer le produit de mise à jour pour cette annonce, l'algorithme semble tout de même assez efficace au vu du nombres de mondes et relations. La Figure 7 présente l'évolution du modèle (partiel).

[5]Les figures présentées dans la section 4.2 proviennent d'un diaporama réalisé par Francois Schwarzentruber (cf. biliographie).

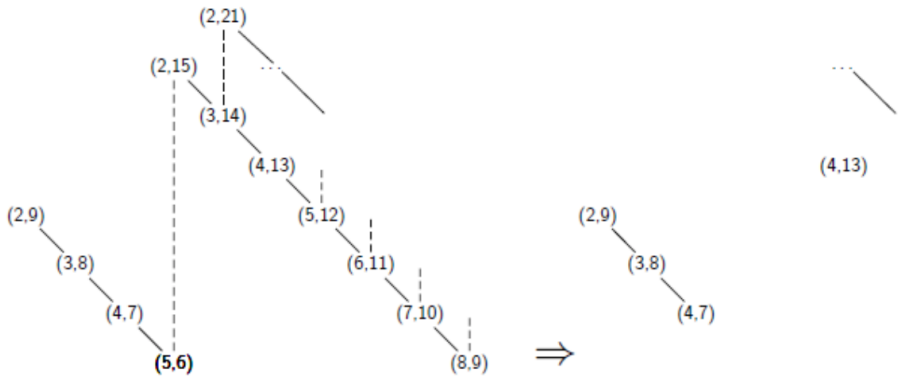


Figure 8: Modèle (partiel) mis à jour après l'annonce de $\bigvee_{x,y}(K_P\ x \vee y)$

Après la deuxième et troisième annonce Lors des deux annonces, le modèle perd drastiquement tous ses mondes possibles jusqu'à n'en avoir plus qu'un : le couple (4, 13). Il s'agit de la solution au problème.

5 Conclusion

Lors de la licence mais aussi en M1 Informatique DECIM cette année, il était souvent question de problèmes de logiques qu'elle soit propositionnelle, modale, ou avec des prédicats... Ce projet m'a alors intéressé car je m'attachais à comprendre ce qu'est la logique et ce que l'on peut en faire et je souhaitais y travailler autour d'un projet. Ce projet annuel était donc une bonne opportunité et ma connaissance sur la logique d'un point de vue général est plus solide grâce à lui. La rédaction d'un rapport permet également de bien synthétiser les choses et de structurer ses acquis.

Les logiques épistémiques permettent de raisonner sur des problèmes simples (par exemple les automobilistes) mais également sur des problèmes peu voir pas du tout intuitifs pour l'homme comme c'est le cas de *The Sum-and-Product Riddle*. J'y ai découvert comment on peut raisonner au travers d'une logique modale avec des modèles en forme de graphes orientés tel que sont les modèles de Kripke. D'autre part, au delà de la logique, cela m'a permit de prendre conscience de l'utilité de représenter des graphes avec les listes de successeurs et/ou des matrices d'adjacences.

Un début de librairie a été créé grâce à ce projet et d'après mes lectures et mes recherches, il y a de la matière pour aller plus loin et réaliser encore plus de fonctionnalités.

References

[1] E. Gillet. La logique de la connaissance, 1993.

[2] F. Schwarzentruber. *Centre International de Rencontres Mathématiques*. Conférence vidéo sur YouTube, `https://www.youtube.com/watch?v=VO_favwnQBA`, 2015.

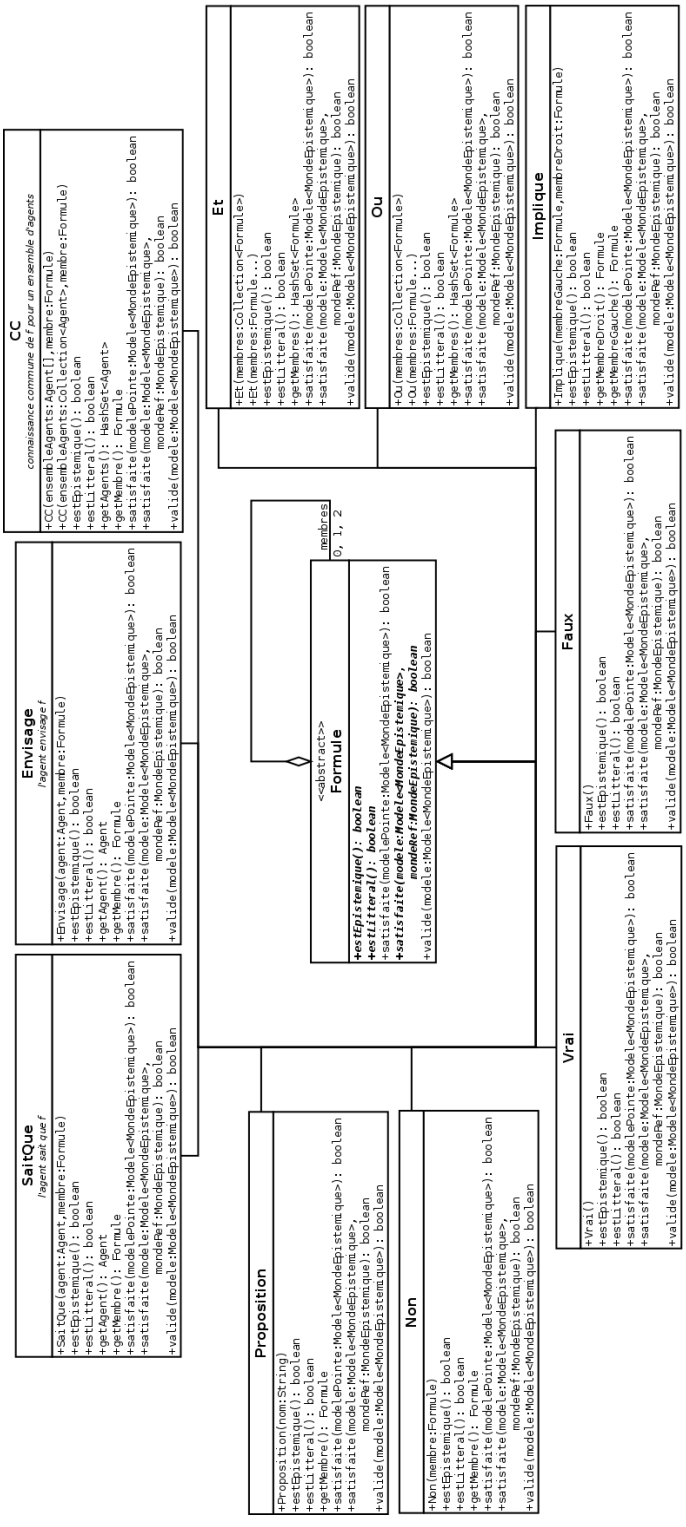
[3] H. P. Van Ditmarsch, J. Ruan, R. Verbrugge. Sum and Product in Dynamic Epistemic Logic, 2008.

[4] J. van Eijick. DEMO (Dynamic Epistemic Modelling) Tool, `http://homepages.cwi.nl/~jve/software/demo_s5/`, 2017.

[5] P. Seban, Who May Say What? Thougths about Objectivity, Group Ability and Permission in Dynamic Epistemic Logic (thesis), 2011.

[6] T. Charrier, F. Schwarzentruber. A Succinct Language for Dynamic Epistemic Logic, 2017.

Annexe A : Diagramme UML du package `formule`



Annexe B : Diagramme UML du package `modeles`