Rapport du Projet d'Application Finale:

[PAF AAI : Une intelligence artificielle qui argumente]

Albane BONNAUD - Justin DALLANT - Guerric DELAGE - Nicolas ROSSET - Haozhe SUN

Du 19 au 29 juin 2017

Glossaire :

- Monde/Perception: le monde regroupe l'ensemble des faits réalisés, des faits qui existent. Dans un cas donné, le monde s'ancre dans une situation initiale fournie à la machine, et est capable de propager la conséquence d'un fait réalisé.

- Nécessité : Indique le poids d'un fait ou d'un souhait. Exemple : je veux une bonne note avec une force 20 mais je n'ai pas envie de travailler avec une force -30, et pire que cela je suis malade avec une force -40.

- Règles : les règles régissent la méthode d'argumentation. Dans un premier temps nous avons considéré les règles causales, qui lient une cause à sa/ses conséquences et réciproquement. Exemple : (je veux une jolie porte) implique (il faut peindre et décaper) et réciproquement en ayant (peint et décapé) j'aurai une (jolie porte).

- Procédure CAN : la procédure CAN constitue le cœur de notre projet puisqu'elle est ici adoptée pour permettre à la machine de bâtir un raisonnement, d'argumenter ! Elle s'appuie sur trois éléments clés : Conflit-Abduction-Négation.

- Conflit : Sans conflit il n'y a pas de dialogue. Ici il ne s'agit pas de créer une ontologie mais bien de mettre en défaut certaines connaissances par rapport à d'autres. Si il y a une conversation c'est qu'il y a un débat possible, matière à discuter. Il y a conflit entre ce que l'on sait et ce que l'on souhaite, entre ce que l'on pense et ce que dit l'autre, etc…De la même façon ici sur des exemples simples, il y a un conflit entre le fait qu'un bricoleur veut une jolie porte mais qu'elle est sale… Il pourrait alors retirer la vieille peinture avec un chalumeau. Mais il reste des morceaux de peinture…Il pourrait alors décaper, etc…

- Abduction : l'abduction est une manière de raisonner qui remonte des conséquences à la cause, afin de trouver une solution. Par exemple comme (repeint) implique (jolie porte), si l'on veut une jolie porte on va remonter à ses causes et donc penser à repeindre.

- Négation : Lorsqu'on remonte à une cause qui n'a elle-même pas de cause, le raisonnement par abduction ne suffit plus. Il faut alors passer la dernière cause à la négative et s'intéresser aux causes de cette négation (donc appliquer de nouveau l'abduction mais sur la négation). Par exemple : je suis arrivée à la conclusion qu'il faut décaper. Décaper n'a pas de cause indiquée. Si l'on rencontre un conflit alors on observe (ne pas décaper) et on trouve que la cause est (ne pas avoir de brosse métallique). Je voulais donc décaper mais je n'ai pas le matériel nécessaire.

- Défaut : Les données par défaut nous donnent ce qui est considéré comme acquis au début de la discussion mais qui ne l'est pas nécessairement dans le monde. Exemple : le bricoleur tient pour acquis que sa porte est en bois dur, mais se rend compte plus tard en la grattant qu'il l'abîme parce qu'elle est en bois tendre. Par défaut il avait pensé à du bois dur, mais dans le monde le bois était tendre.

- Pré-requis : Les pré-requis sont ce qui conditionnent la réalisation d'une action. Exemple : avoir le matériel nécessaire, comme la brosse métallique pour décaper.

Rapport narratif

Une fois données les explications de Monsieur Dessales sur les bases de l'argumentation avec la notion de règles, de nécessité et de perception, nous avons cherché à modéliser un exemple simple. Cet exemple est basé sur une conversation basique de bricoleurs à propos de la peinture des portes. Le but étant d'obtenir une jolie porte, la conversation évolue suivant les difficultés rencontrées (par exemple une surface non lisse) et les solutions proposées (décaper la porte).

Premier modèle prévu:

Une classe unique "Facts" regroupant les actions, les effets, et les données par défaut.

Les objets de cette classe sont liés entre eux par des liens logiques (en l'occurence causaux uniquement: nice_surface ==> burn_off)

Chaque objet peut aller chercher ses causes et ses conséquences.

Ecriture d'une procédure argumentative qui manipule ces objets (méthode d'abduction, de négation, etc...).

Problème rencontré: pas réaliste au niveau du code

Premier modèle codé:

5 classes:

- *Preprocessing*: extrait d'une conversation textuelle l'ensemble des règles

- *Predicate*: regroupe les faits, actions etc...

- *Logical_Links*: construit des objets "liens logiques" pouvant relier des prédicats entre eux

- *Causal_Links*: sous classe des liens logiques, qui sont ici exclusivement causaux. Relient un prédicat à ses causes et ses conséquences. Peut propager lui-même la réalisation d'une cause à ses conséquences.

- *Argumentator*: met en place la procédure CAN (causalité, abduction, négation)

Problème rencontré: comment est représenté le monde ?

Seconde version du code:

Ajout d'une sixième classe World, qui sépare le monde de la connaissance pure (c'est-à-dire de l'ensemble des règles). Cette classe dispose d'une méthode qui effectue le "chaînage avant" des conséquences à partir d'une cause réalisée.

Mais il était également nécessaire de s'assurer qu'une conséquence restait vraie si au moins une de ses causes suffisantes était vraie, et de même de s'assurer qu'une conséquence n'entrait pas en contradiction avec une donnée immuable du monde, pour ce faire la fonction implémentait un retour en arrière à partir des conséquences vers les divers liens logiques dans lesquels elles étaient impliquées. L'état des causes était vérifié (Sont-elles réalisées ou non ?)

Problème rencontré: représentation du monde incomplète (le monde n'argumente pas donc on est de toutes les façons obligés de tricher sur la façon dont les faits s'enchaînent mais ici le retour en arrière sur toutes les causes de chaque

conséquence pouvait occasionner des incohérences et des décalages si de nouvelles actions en cours parallèlement n'étaient pas prises en compte).

Troisième version du code:

Lorsqu'une cause est réalisée toutes les conséquences de liens logiques (hormis ceux dans lesquels cette cause est impliquée) sont remises à l'état initial. Ensuite on applique sans souci la conséquence de cette cause, conséquences qui peuvent de nouveau avoir des causes donc on boucle et on applique les effets jusqu'à ce que plus rien ne change.

Problème rencontré: notre modélisation ne prend pas encore en compte les pré-requis des actions (exemple: pour décaper il faut un chalumeau...)

Quatrième version modèle du code:

Prise en compte des pré-requis sous forme négative: non(brosse métallique) ==> non(décaper). Donc, quand une action n'a pas de cause et est passée à la négation, si elle n'est pas faisable, cela indique que l'on n'a pas d'outil nécessaire.

+ retour sur la conception du monde: retour à la précédente version, car il n'est pas possible de vouloir remettre les causes à zéro.

Cinquième version du code:

A partir d'un nouvel exemple, création d'un nouveau type de lien logique (l'incompatibilité). En effet l'exemple traité était sur la place des femmes au tennis. Pourquoi jouent-elles en 3 sets et pas en 5 sets ? Il fallait alors considérer des évènements incompatibles pour raisonner comme: (jouer en 3.sets) et (jouer en 5.sets).

En parallèle: Réalisation du POSTER, travail sur les prédicats avec variables, et construction d'un nouvel exemple de dialogue sur la Proportionnelle en politique.

La prise en compte des variables a posé plusieurs problèmes. En premier lieu il a fallu adapter le préprocesseur et reconcevoir la classe predicate, puisque pour un objet de cette classe correspondent plusieurs prédicats (exemple l'objet (joue) peut prendre plusieurs arguments et permettre d'obtenir les prédicats (les hommes jouent en 5 sets) ou (les femmes jouent en 3 sets)). La structure générale du programme ne changeait

pas mais nous avons été confronté à deux problèmes majeurs que nous n’avons pas eu le temps de résoudre : l’unification au niveau des règles pour s’assurer de la correspondance des variables à droite et à gauche, et d’autre part le stockage de la caractéristique (realised) car il était impensable de stocker toutes les combinaisons de valeurs possibles pour chaque prédicat, surtout dans une optique de passage à l’échelle.

Sixième et dernière version du code:

Intégration du langage naturel pour progresser vers une conversation réaliste, et donc à terme vers une interaction réelle avec un interlocuteur.

En parallèle: Travail sur l'interface graphique, puis sur la démo attendue en fin de semaine.