ROMON Pierre
FIERS Charles

TP3
Mise en œuvre d'un SE d'ordre 0+

Table des matières

Représentation Lisp

Base de règles

On cherche tout d'abord à implémenter notre base de règles, chose nécessaire au fonctionnement du moteur d'inférence. Notre base de règles, forte de 66 règles, est construite comme une liste de listes, chaque sous-listes étant une règle. Pour définir une règle, on commence par une clé unique pour chaque règle. L'élément suivant d'une règle est une liste contenant les prémices. Enfin, le but est le dernier élément de la liste. On choisit cette construction afin de naviguer facilement dans la liste de règles en utilisant les clés. Grâce au prédicat *assoc*, on peut récupérer les prémisses ou le but d'une règle.

Base de faits

On instancie chaque fait comme un couple (attribut, valeur). On utilise donc une liste qui contient successivement l'attribut et la valeur du fait. En effet, nous utilisons ici un moteur d'inférence 0+ où chaque fait est défini par un couple (attribut, valeur). On peut différencier les faits dans plusieurs catégories : faits initiaux, faits intermédiaire et faits classiques.

D'abord les faits initiaux.
boisson = {eau, pampryl, vin, biere}
caractere = {humble, sage, brave, ambitieux}
hobby = {reve, sport, boire, forniquer}
paysage = {ville, campagne, mer, montagne}
arme = {epee, arc, magie, laser}

Nous avons ici les faits solutions.
maison = {targaryen, stark, lannister, tyrel, greyjoy, baratheon, martel, tully, aryn}
La Garde de Nuit est un état qui est atteint quand aucune maison ne correspond puisque ses membres n'appartiennent à aucune maison.

Et voilà enfin les faits intermédiaires permettant le raisonnement. Les faits ici définis sont des faits fictifs issus de l'univers de Game of Thrones.
valeur = {honneur, fierte, famille, pouvoir}
religion = {nouveau, ancien, rouge, sel}
boussole = {nord, sud, est, ouest}
fortune = {riche, pauvre}
armee = {mercenaire, banneret, dragon, bateau}
strategie = {manipulation, guerre, neutre, sympathie}
objectif = {regne, vengeance, protection, conquete}
cheveu = {brun, blond, roux}
ordre = {chevalier, mestre, pretre,

Moteur d'inférence

Fonctions de service

On définit plusieurs fonctions de service pour faciliter l'implémentation des moteurs d'inférence.

La première d'entre elles est la fonction spot. Cette fonction prend deux arguments : un couple (attribut, valeur) item et une liste de couples liste. Globalement cette fonction remplit le rôle du prédicat *member* mais pour des couples (attributs, valeur) et non pour des éléments atomiques. Pour cela, on parcourt la liste et pour chaque élément, on vérifie l'égalité entre le premier et le second élément. On retourne l'élément cherché s'il est bien dans la liste, sinon on retourne *nil*.

On utilise ensuite la fonction *listeRegles*. Cette fonction ne prend aucun argument et retourne la liste des clés des règles de la base de règles. Pour cela, on a recours à au prédicat *mapcar*.

La fonction *premisse* prend la clé d'une règle en argument et retourne la liste des prémisses correspondante. Un simple *cadr* associé à un *assoc* est suffisant.

Quant à la fonction but, elle prend elle-aussi en argument la clé d'une règle et retourne le but de règle correspondante. On utilise simplement un *caddr* et un *assoc*.

Chaînage avant en largeur d'abord

À chaque récurrence du moteur, on commence par établir les règles candidates selon la base de faits avec *reglesCandidatesAvant*. Pour cela, on parcourt la liste des clés des règles. Pour chaque règle, on vérifie si les prémisses de la règle sont bien présentes dans la base de faits grâce à la fonction *verifPremissesAvant* et que le but n'y est pas déjà présent. Ces deux conditions respectées, on ajoute la règle à la liste des clés candidates. On met ensuite à jour la base de faits en y ajoutant les buts des règles candidates avec la fonction *majBF*. On arrête la récurrence lorsque l'on est capable de déterminer l'appartenance à une maison. Pour cela, on cherche dans la base de fait un fait appartenant aux états finaux avec la fonction *allegeanceAvant*. Dans ce cas-là, on indique la maison correspondante. Dans le cas où le raisonnement est bloqué car il n'y a plus de règles candidates possibles, on assigne la garde de la nuit comme maison. Le moteur est relativement rapide.

Chaînage arrière en largeur d'abord

Pour le chaînage arrière, on met en place une base de buts qui stockera les buts successifs. Dans ce moteur, une règle est candidate si au moins un de ses prémisses n'est pas encore dans la base de buts (vérification avec *verifPremissesArriere*) et si son but est déjà présent dans cette même base. Pour mettre à jour la base de buts, on y ajoute les prémisses des règles candidates qui ne sont pas encore présent dans la base buts grâce à *majBB*. Pour arrêter la récurrence, on vérifie l'appartenance à une maison qu'on affiche. On peut également sortir de la récurrence quand le raisonnement est dans une impasse, c'est à dire qu'il n'y a plus de règles candidates. On réitère ce processus pour chaque état finaux grâce à un *dolist*. Si aucune maison ne correspond, on affecte à la Garde de nuit. On remarque que ce moteur est bien plus lent à l'exécution, chose normale puisqu'il y a de nombreux états finaux.

Scénarios

Scénario 1

Pour le premier scénario, nous avons choisi de faire une démonstration du moteur d'inférence en chaînage avant. On prend le cas d'un citoyen de Westeros qui serait brave, armé d'une épée, vivant au bord de la mer, amateur de sport et friand de bière. Le moteur retourne la maison Greyjoy, un archipel peuplé de fiers guerriers et marins réputés pour être des buveurs chevronnés.

Scénario 2

C'est le moteur d'inférence en chaînage arrière qui fonctionne pour ce scénario. On prend une personne sage vivant dans les montagnes, imaginatif, armée d'un arc et amateur de vin. D'après notre programme, cette personne pourrait appartenir à la maison Targaryen, Lannister ou Tyrel. Résultat logique puisque ces riches maisons amatrices de vins sont très ambitieuses.

Scénario 3

Encore une fois, on utilise le moteur en chaînage avant puisque celui-ci est bien plus optimisé. En essayant de prendre des caractéristiques très contradictoires (vin, magie sage, ville, boire), on obtient ainsi la garde de la nuit. Ce résultat est normal car avec des caractéristiques aussi hétérogènes, aucune maison ne peut correspondre.