République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université de Jijel
Faculté des Sciences Exactes et Informatique
Département d'Informatique

MII IA

SYSTEMES INTELLIGENTS

Support de cours élaboré par Dr. Saloua CHETTIBI Maître de Conférences B en Informatique | 2019/2020

Avant-propos

Le présent support de cours est le fruit de cinq ans d'expérience dans l'enseignement de la matière intitulée « ***Systèmes Intelligents*** » aux étudiants en master II spécialité « ***Intelligence Artificielle***».

« *Développer un système intelligent* » est évidemment la question principale en IA. Mais que désigne-t-on par *système intelligent* ? Les bonnes réponses sont multiples : « c'est un système qui raisonne, qui supporte l'incertitude, qui est doté de la faculté d'apprentissage[1] … la réponse peut être aussi : c'est un système qui peut traiter un langage naturel».

En effet, l'intelligence est multi-facette. C'est pourquoi les chapitres dans ce support de cours semblent à première vue un peu hétérogènes. Les thématiques suivantes sont abordées : la conception des agents intelligents, les ontologies, les logiques non classiques, à savoir : les logiques de description, la logique floue, les logiques modales et temporelles. Une introduction au Traitement Automatique du Langage Naturel (TALN) est également prévue.

L'*objectif pédagogique* ultime de la matière SI est de faire découvrir aux étudiants comment ces différentes thématiques sont en fait étroitement liées :

- Un système intelligent peut être mono-agent ou multi-agent (SMA).
- Un agent intelligent peut mener un raisonnement approximatif via un système d'inférence floue.
- Le comportement désiré d'un agent peut être spécifié/vérifié grâce aux logiques temporelles.
- Les différents agents dans un SMA peuvent communiquer via un vocabulaire défini par le biais d'une ontologie.
- Une ontologie est un modèle de représentation de connaissance souvent formalisé en logique de description.
- Une ontologie peut être construite à l'aide des outils du TALN.
- La représentation de la sémantique d'un langage naturel s'appuie sur les différents formalismes de représentation des connaissances y compris les ontologies.
- …etc.

La matière est très intéressante et n'a pour prérequis que la connaissance des logiques des propositions et des prédicats. Le contenu de ce document est toujours en évolution : toute remarque susceptible d'améliorer sa qualité est la bienvenue.

L'auteur, Novembre 2019.

[1] La partie de l'apprentissage est volontairement omise puisque elle fait l'objet d'un autre module dans le cursus IA.

Table des Matières

Liste des Figures

Liste des Tableaux

CHAPITRE 1

Agents Intelligents

1.1 Introduction

L'Intelligence Artificielle (IA) peut être définie comme l'étude de la conception d'agents intelligents [1], c'est-à-dire des systèmes qui montrent un comportement rationnel. Un système est dit rationnel s'il fonctionne correctement compte tenu de ce qu'il sait. Cette approche de l'IA est au cœur de ce premier chapitre centré sur le thème de la conception des agents intelligents.

Ce cours récapitule les caractéristiques d'un agent intelligent et met surtout l'accent sur le compromis « réactivité/proactivité ». La conception d'un agent intelligent passe d'abord par la spécification de son environnement de tâche et ensuite par le choix d'une architecture convenable (réactive, délibérative ou hybride). Mais avant d'aborder le comment ? (i.e. les architectures concrètes), les étudiants doivent comprendre de quoi s'agit-il ? En réponse à cette question, le modèle abstrait des agents est développé.

Il faut souligner que les exemples utilisés dans le chapitre portent en général sur des robots mobiles. La ***robotique mobile*** étant un domaine d'application typique pour les agents intelligents.

1.2 Concept d'Agent en IA

Le mot **Agent** a pour origine le terme latin « ***agere*** » qui signifie « ***faire*** ». Simplement, un Agent peut être vu comme une « ***une entité qui agit*** ». Cette entité peut être physique ou virtuelle [2]: une entité physique est quelque chose qui agit dans le monde réel (robot, avion, voiture). En revanche, un composant logiciel, un module informatique sont des entités virtuelles car elles n'existent pas physiquement.

Une définition plus élaborée de ce qui est un agent est la suivante [1]: « ***Un agent est tout ce qui peut être vu comme percevant son environnement au travers de capteurs et agissant sur cet environnement au travers d'effecteurs***». Un agent générique ou standard peut être schématisé comme dans la figure ci-dessous.

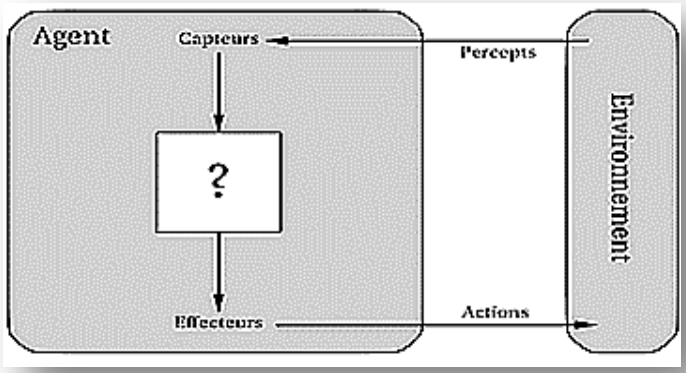


Figure 1.1 Schéma d'un agent standard [1].

Sujet de Débat

Peut-on considérer un ***système expert*** (de genre MYCIN) comme Agent ?
Un tel système n'interagit pas directement avec un environnement:

- il reçoit ses informations non pas via des capteurs mais par l'intermédiaire d'un utilisateur agissant comme intermédiaire.
- il n'agit pas sur un environnement mais donne des commentaires ou des conseils à un tiers.

La réponse doit être alors NON [3].

NB. Mycin est un système Expert (développé dans les années 1970) servant au diagnostic des maladies infectieuses.

1.3 Caractéristiques d'un agent intelligent

Il est très difficile de donner une définition finale pour « l'Intelligence » et c'est également le cas si on cherche à définir ce qui est « un agent intelligent ». Selon Frékon et Kazar [4], un agent est dit **intelligent** s'il est : ***intentionnel, cognitif, rationnel et adaptatif***.

- L'***agent intentionnel*** possède des buts et des plans explicites lui permettant d'accomplir ses buts [2].

- L'***agent cognitif*** est un agent qui mène des raisonnements afin de choisir ses actions [4]. Il est doté d'une représentation interne de son environnement avec un mécanisme d'inférence.
- *L'**agent rationnel*** est celui qui choisit toujours l'action appropriée. Plus formellement, un agent rationnel choisit l'action qui maximise l'espérance d'une ***mesure de performance [1]***.
- L'***agent adaptatif*** est un agent capable de moduler son comportement selon l'état de l'environnement [4]. Autrement, c'est un agent doté de la faculté d'apprentissage qui lui permet de compenser ses connaissances a priori incomplètes voire erronées [1].

Autres propriétés désirables des agents intelligents [3]:

- ***Autonomie***: l'agent est plus ou moins indépendant de l'utilisateur et des autres agents.
- ***Sociabilité***: capacité d'interagir avec d'autres agents ou même avec des humains en utilisant un ACL (*Agent Communication Language*).
- ***Réactivité***: capacité de répondre rapidement aux changements dans l'environnement.
- ***Proactivité***: capacité de prendre l'initiative c'est-à-dire prévoir les évènements futurs et donc d'anticiper en planifiant les actions à accomplir.

Un agent est qualifié de ***flexible*** s'il vérifie les trois dernières propriétés.

Sujet de débat

Réactivité Vs Proactivité avec un exemple adapté de la référence [2] :

Imaginons un robot qui veut franchir une porte et que celle-ci se ferme à clef. S'il s'agit d'un ***agent proactif***, il pourra construire (d'une certaine manière) un plan « ouvrir Porte» :

> Aller jusqu'à l'endroit où se trouve la clef
> Prendre la clef
> Aller jusqu'à la porte
> Ouvrir la porte avec la clef

S'il s'agit d'un ***agent réactif***, on peut construire les règles de fonctionnement suivantes :

> **Si** je suis devant la porte et que j'ai une clef **Alors** l'ouvrir
> **Si** je suis devant la porte et sans clef **Alors** essayer de l'ouvrir
> **Si** la porte ne s'ouvre pas et que je n'ai pas la clef **Alors** aller chercher la clef
> **Si** je cherche une clef et qu'il y a une clef devant moi **Alors** prendre la clef et aller vers la porte

Comparons les deux agents :

Proactif	***Réactif***
Le plan est construit par l'agent lui-même.	Les règles sont données explicitement par le concepteur.
Point fort : Optimisation (moins de déplacements si la porte est fermée)	Point fort : Souplesse (si la porte est ouverte, l'agent ouvre directement sans aller chercher la clef)

Bien que la distinction entre réactivité et proactivité soit utile dans un premier temps, mais il ne faut pas comprendre qu'il s'agit d'une opposition catégorique. Bien au contraire, l'enjeu actuel est de réaliser des agents qui mêlent des capacités cognitives et réactives [2].

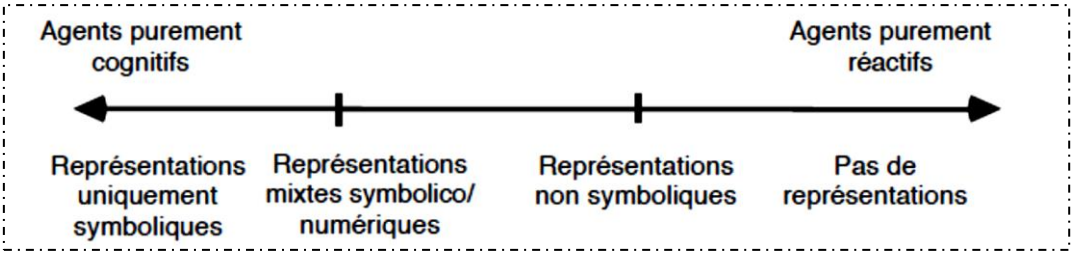


Figure 1.2 La distinction cognitif/réactif (Axe pratique d'évaluation de la capacité des agents à accomplir individuellement des tâches complexes et à planifier leurs actions) [2].

1.4 Structure des agents intelligents

La structure d'un agent intelligent peut être décrite comme suit [1] :

Agent* = *Architecture* + *Programme

Le **programme agent** implémente la fonction associant aux perceptions des actions (*la boite noire dans la Figure 1.1*). L'***architecture*** quant à elle désigne un équipement informatique doté d'effecteurs et de capteurs physiques. Elle transmet au programme les perceptions issues des capteurs, exécute le programme et achemine les actions choisies aux effecteurs. Nous nous intéressons dans ce qui suit à la ***conception des programmes agents***.

Selon le fonctionnement du programme agent, on distingue les quatre types d'agents suivants [1]:

a) Agent réflexe simple

Il sélectionne des actions en fonction de la perception courante et ignore le reste de l'historique des perceptions. Le comportement d'un agent réactif est régi par un ensemble de règles de la forme « Si situation(s) ou condition(s) Alors action ».

Exemple : le thermostat d'un chauffage est muni d'un capteur pour détecter la température ambiante. Selon si la température est actuellement trop basse ou bonne, une action est choisie : « allumer chauffage » ou « éteindre chauffage».

b) Agent réflexe fondé sur des modèles

Ce type d'agent doit maintenir des informations concernant : i) la manière dont le monde évolue indépendamment de l'agent et ii) la façon dont les actions de l'agent affectent le monde. La perception courante est combinée avec l'ancien état interne pour mettre à jour l'état courant en se basant sur le modèle interne.

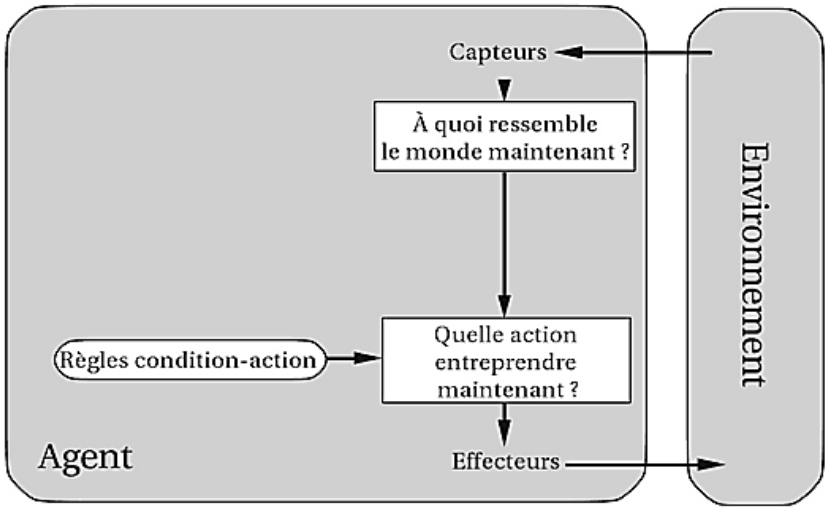


Figure 1.3 Représentation schématique d'un agent réflexe simple [1]

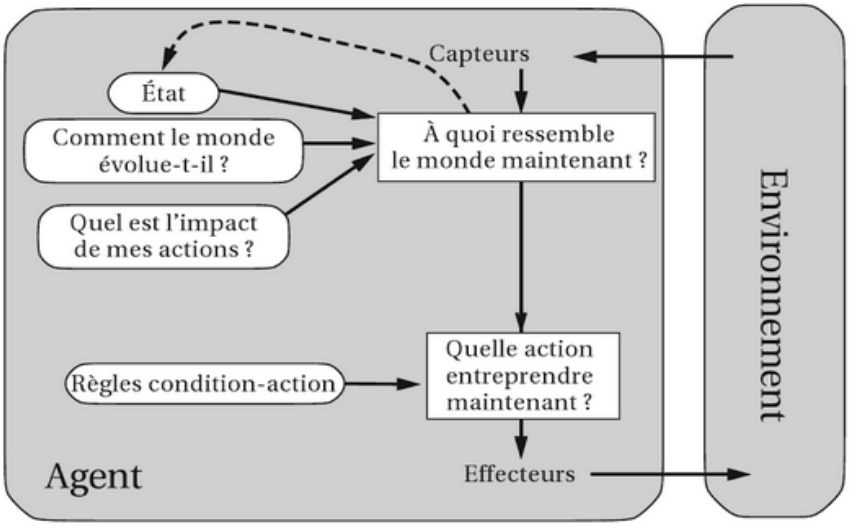


Figure 1.4 Représentation schématique d'un agent basé sur un modèle [1].

c) Agent fondé sur les buts

Dans la sélection de ses actions, ce type d'agent est guidé non seulement par l'état actuel de l'environnement mais également par son but final.

d) Agent fondé sur l'utilité

Il est possible d'atteindre ses buts en suivant différentes séquences d'actions, mais quelle est la meilleure séquence : la plus sûre, la moins coûteuse, la plus rapide, etc. ? L'agent doit choisir les actions susceptibles de maximiser sa fonction d'utilité.

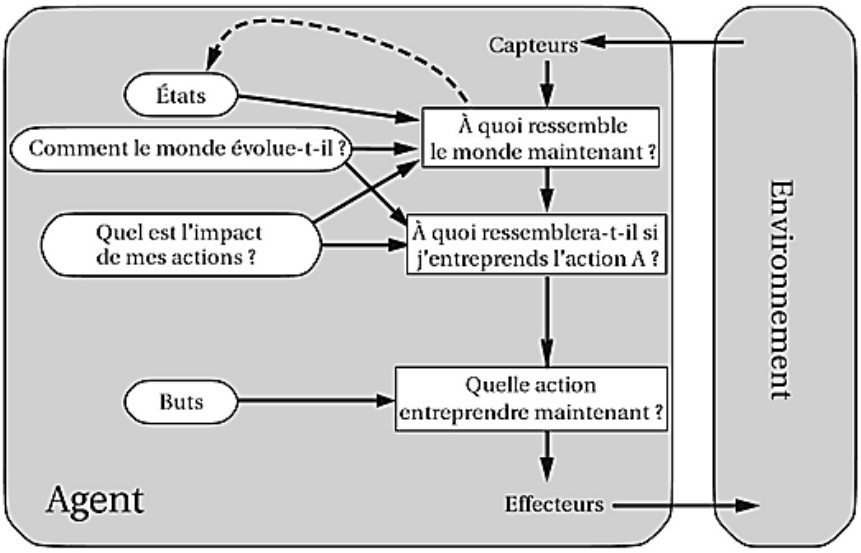


Figure 1.5 Représentation schématique d'un agent basé sur les Buts [1].

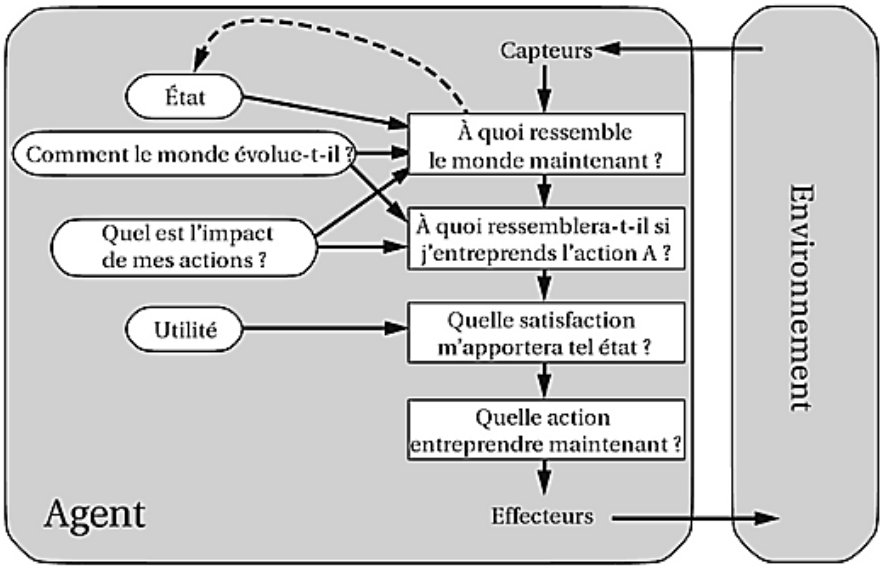


Figure 1.6 Représentation schématique d'un agent basé sur l'utilité [1].

1.5 Environnement de la tâche et ses propriétés

L'environnement est l'univers dans lequel l'agent évolue et effectue ses tâches. L'environnement de la tâche est défini par la combinaison : mesure de performance, environnement, effecteurs et capteurs (PEAS en anglais de : *Performance Environment Actuators Sensors*).

Avant de construire un agent intelligent, on doit spécifier son PEAS. Dans le Tableau 1.1 se trouvent des exemples d'agents avec leurs spécifications PEAS.
On distingue différents types d'environnements, en l'occurrence [1]:

- Entièrement observable *(Vs partiellement observable):* si l'agent peut avoir une information précise, complète et à jour concernant l'état de l'environnement alors l'environnement est entièrement observable. *(à l'inverse : données bruités, incomplètes ou imprécises…..)*

- Mono-agent (Vs multi-agent : concurrentiel ou coopératif).
- Déterministe *(Vs stochastique) :* si l'état suivant de l'environnement est complètement déterminé par l'état actuel et par l'action exécutée alors l'environnement est déterministe, il est stochastique sinon.
- Séquentiel *(Vs épisodique):* dans les environnements séquentiels, la décision courante est susceptible d'affecter les décisions futures. *(In contrario, chaque épisode est indépendant).*
- Statique *(Vs dynamique):* l'état de l'environnement ne change pas au cours de la prise de décision. Soulignons que le monde physique est un environnement hautement dynamique.
- Discret *(Vs continu)* : s'il existe un nombre fini d'actions et de perceptions alors l'environnement est discret.

Le Tableau 1.2 récapitule les propriétés de plusieurs environnements familiers.

Type d'agent	Mesure de performance	Environnement	Effecteurs	Capteurs
Système de diagnostic médical	Rétablissement des patients, minimisation des coûts	Patient, hôpital, personnel	Affichage de questions, tests, diagnostics, traitements, orientation	Entrée au clavier de symptômes, recherches, réponses du patient
Système d'analyse d'images satellites	Catégorisation correcte des images	Liaison descendante depuis un satellite en orbite	Affichage de la catégorisation de la scène	Tableaux de pixels colorés
Robot contrôleur de pièces	Pourcentage de pièces dans les bonnes corbeilles	Tapis roulant avec pièces, corbeilles	Bras et main articulés	Caméra, capteurs articulés
Contrôleur de raffinerie	Pureté, production, sécurité	Raffinerie, opérateurs	Valves, pompes, réchauffeurs, affichages	Capteurs de température, de pression et chimiques
Répétiteur d'anglais interactif	Notes des étudiants aux contrôles	Ensemble d'étudiants, organisme faisant passer les tests	Affichages d'exercices, de suggestions, de corrections	Entrées au clavier

Tableau 1.1 Exemples d'agents et de leurs spécifications PEAS [1].

Environnement de tâche	Observable	Agents	Déterministe	Épisodique	Statique	Discret
Problème de mots croisés	Entièrement	Mono	Déterministe	Séquentiel	Statique	Discret
Partie d'échecs chronométrée	Entièrement	Multi	Déterministe	Séquentiel	Semi	Discret
Poker	Partiellement	Multi	Stochastique	Séquentiel	Statique	Discret
Backgammon	Entièrement	Multi	Stochastique	Séquentiel	Statique	Discret
Conduite de taxi	Partiellement	Multi	Stochastique	Séquentiel	Dynamique	Continu
Diagnostic médical	Partiellement	Mono	Stochastique	Séquentiel	Dynamique	Continu
Analyse d'images	Entièrement	Mono	Déterministe	Épisodique	Semi	Continu
Robot détecteur de défauts	Partiellement	Mono	Stochastique	Épisodique	Dynamique	Continu
Contrôleur de raffinerie	Partiellement	Mono	Stochastique	Séquentiel	Dynamique	Continu
Répétiteur d'anglais interactif	Partiellement	Multi	Stochastique	Séquentiel	Dynamique	Discret

Tableau 1.2 : Exemples d'environnements de tâche avec leurs propriétés [1].

1.6 Modèle abstrait des agents

Le modèle abstrait des agents présenté ci-après est pris de la référence [3]. Commençons par la modélisation de l'***agent générique*** déjà vu dans la section 1.2, pour ce faire considérons:

$\boldsymbol{S} = \{\boldsymbol{s1}, \boldsymbol{s2}, \ldots\}$: L'ensemble des états de l'environnement.

$\boldsymbol{A} = \{\boldsymbol{a1}, \boldsymbol{a2}, \ldots\}$: L'ensemble des actions potentielles.

Alors l'agent peut être vu comme une fonction qui associe à une séquence d'états ***(historique ou expérience)*** une action :

$$\textbf{Action : S}^* \rightarrow \textbf{A}$$

Le comportement ***non déterministe*** d'un environnement peut être décrit par une fonction :

$$\boldsymbol{Env} : \boldsymbol{S\ x\ A} \rightarrow \wp(\boldsymbol{S}).$$

Si l'environnement est **déterministe** alors Env devient :

$$\boldsymbol{Env} : \boldsymbol{S\ x\ A} \rightarrow \boldsymbol{S}$$

L'historique de l'interaction de l'agent avec son environnement, considérée continue, est vu comme une séquence h infinie (S_0 *étant l'état initial*):

$$\boldsymbol{h}: \boldsymbol{S_0} \xrightarrow{\boldsymbol{a_0}} \boldsymbol{S_1} \xrightarrow{\boldsymbol{a_1}} \boldsymbol{S_2} \ldots \xrightarrow{\boldsymbol{a_{u-1}}} \boldsymbol{S_u} \xrightarrow{\boldsymbol{a_u}} \ldots$$

Un agent ***purement réactif*** ne tient pas compte de son expérience passée dans la prise de décision : il réagit simplement à l'état courant du système. Dans ce cas, la fonction action devient :

$$\textbf{Action : S} \rightarrow \textbf{A}$$

Exemple (Un modèle abstrait du thermostat)
Notation utilisée :

s1 (Température - OK) a1 (Allumer chauffage)
s2 (Basse Température) a2 (Eteindre chauffage)

$S = \{s1, s2\}$
$A = \{a1, a2\}$
La fonction Action est alors définie comme suit :

$$\text{Action (s)} = \begin{cases} \text{a2} & \text{si s = s1} \\ \text{a1} & \text{si s = s2} \end{cases}$$

Nous raffinons maintenant la définition de la fonction **Action** en créant la fonction see :

$$\boldsymbol{See : S \rightarrow P}$$

Cette fonction a pour rôle d'associer à un état de l'environnement des perceptions (P). Alors la fonction Action devient :

$$\textbf{Action : P* } \rightarrow \textbf{ A}$$

NB. Pour deux états différents de l'environnement, la fonction ***See*** peut associer la même perception.
Exemple (Implémentation de la fonction See [3])

- *Pour un robot mobile* : une caméra vidéo, un capteur infra rouge, …etc.
- *Pour un agent logiciel* : par exemple des commandes linux comme : ls qui renvoie la liste des répertoires ou fichiers se trouvant dans le répertoire courant et ps qui renvoie la liste des processus en cours d'exécution.

Certains agents maintiennent **un état interne** mémorisant leurs historiques. On définit alors **I** l'ensemble de tous les états internes. **See** ne va pas changer mais Action devient :

$$\textbf{Action : I} \rightarrow \textbf{ A}$$

Dans ce cas, on aura besoin d'une nouvelle fonction qui met à jour l'état interne en fonction de la nouvelle perception:

$$\textbf{Next : I x P} \rightarrow \textbf{ I}$$

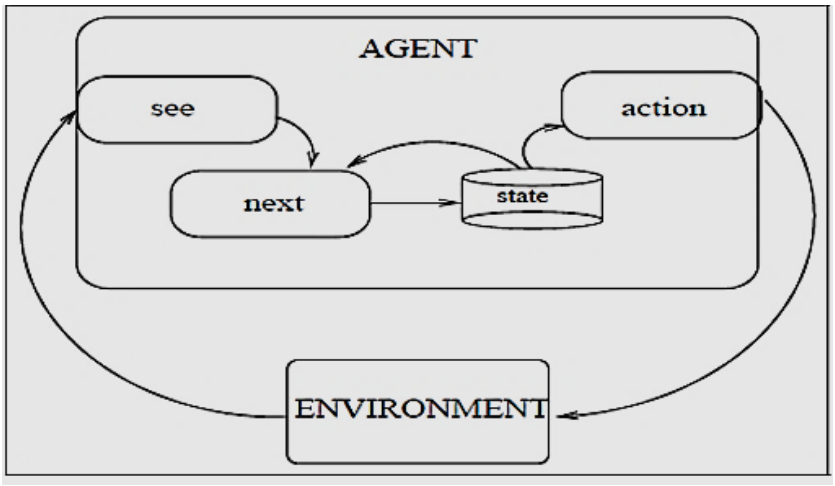


Figure 1.7 Schéma associé au modèle abstrait construit [3].

En résumé :

- l'agent se trouve dans un état interne $\boldsymbol{i_t}$;
- l'agent observe l'état courant de l'environnement $\boldsymbol{S_t}$
- il génère une perception $See(\boldsymbol{S_t})$
- l'état interne est mis à jour par **Next** et devient **Next**$(\boldsymbol{i_t}, \boldsymbol{See}(\boldsymbol{S_t}))$
- Enfin, l'action choisie est : **Action (Next**$(\boldsymbol{i_t}, \boldsymbol{See}(\boldsymbol{S_t}))$)

1.7 Architectures concrètes des agents

Dans la section précédente, nous n'avons pas déterminé à quoi ressemble l'état interne de l'agent ni comment la fonction action est implémentée. En effet, les principales architectures concrètes des agents [3] répondant à toutes ces questions sont décrites dans les sous-sections suivantes.

1.7.1 Agent basé sur la logique

L'approche traditionnelle **de l'IA symbolique** suggère que le comportement intelligent peut être généré par un système si ce dernier est doté d'une représentation symbolique de l'environnement et de celle du comportement envisagé avec une manipulation syntaxique de ces représentations. Dans notre contexte: les représentations symboliques sont faites à l'aide de formules logiques et la manipulation syntaxique correspond à une déduction logique (ou à une preuve de théorème).

Au fait, *l'état interne* d'un agent basé sur la logique correspond à une base de formules logiques notée Δ (i.e. *base de connaissance ou base des faits de l'agent*)**.** Soit L l'ensemble de toutes les formules logiques et soit $D = \wp(L)$ l'ensemble de toutes les bases de connaissance possibles. Alors :

- La fonction **See** reste inchangée.
- La fonction **Next** devient : **Next :D x P → D**
- Pour la fonction Action : D → A, on donne le pseudo code suivant [3] :

```
1.  Fonction action (Δ: D): A
2.  Début
3.      Pour chaque a ∈ A faire
4.          Si  Δ⊢ρ  faire (a) alors
5.              Retourner a
6.          Fin-si
7.      Fin-pour
8.      Pour chaque a ∈ A faire
9.          Si  ¬(Δ⊢ρ  ¬faire (a)) alors
10.             Retourner a
11.         Fin-si
12.     Fin-pour
13. Renvoyer null
14. Fin fonction action
```

- L'état du système sous forme de formules logiques est passé à la fonction qui doit renvoyer une action (Ligne1).

- L'agent cherche une action qui peut être dérivée avec les règles de déduction $\boldsymbol{\rho}$. Si une action est trouvée alors elle est renvoyée en sortie (Ligne 5).
- Si aucune action n'est trouvée, l'agent cherche au moins une action qui ne soit pas explicitement interdite c'est à dire pour laquelle on ne peut pas dériver $\neg\, faire\,(a)$ (Ligne 10).
- Si les tests précédents échouent, alors l'action Null (rien faire) est retournée (Ligne 13).

Exemple [3]

Considérons l'exemple classique d'un ***robot aspirateur*** dont l'objectif est de nettoyer une pièce (représentée par une grille 3x3). Initialement, l'agent est dans la position (0,0) en face du nord. Seulement les déplacements illustrés sur la figure sont autorisés (autrement dit : la trajectoire du robot est prédéfinie):

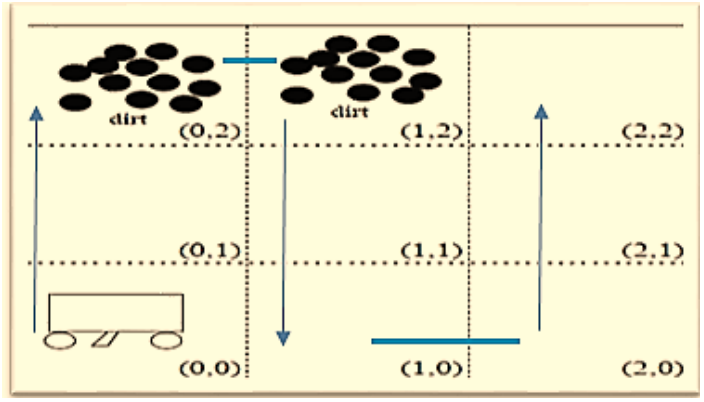


Figure 1.8 L'environnement simplifié du robot aspirateur.

Nous considérons que l'aspirateur est doté d'un capteur approprié pour la détection de la saleté. Ce capteur permet l'acquisition de l'état de l'environnement. On suppose l'existence d'une certaine fonction See permettant la transformation des états aux perceptions suivantes: $P = \{S, X, Y, D\}$

Tel que : S désigne Saleté, D désigne direction, (X,Y) représente la position de l'aspirateur.

L'ensemble des actions potentielles est : $A = \{Avancer,\ Nettoyer, Tourner\}$.

Remarque : Le raffinement de l'action Tourner (à gauche ou à droite ?) est laissé aux étudiants.

En outre, définissons les prédicats du domaine :

$\boldsymbol{L(x,y)}$: l'agent se trouve dans la localisation (x,y)

$\boldsymbol{S(x,y)}$: présence de la Saleté dans la position (x,y)

$\boldsymbol{F(d)}$: l'agent est en face de la direction d (Nord, Est, Ouest ou Sud).

L'aspirateur se trouve dans l'état initial (ce que l'agent connait sur son environnement) :

$$\Delta_0 = \{L(0,0), \neg S(0,0), F(Nord)\}$$

Les règles de déduction qui régissent le fonctionnement de l'agent aspirateur sont les suivantes:

$\boldsymbol{L(x,y) \wedge S(x,y) \vdash\ faire\,(Nettoyer)}$

Cette règle est prioritaire aux règles de navigation suivantes:

$$\boldsymbol{L(0,0) \wedge F(N) \wedge \neg S(0,0) \vdash faire\ (Avancer)}$$
$$\boldsymbol{L(0,1) \wedge F(N) \wedge \neg S(0,1) \vdash faire\ (Avancer)}$$
$$\boldsymbol{L(0,2) \wedge F(N) \wedge \neg S(0,2) \vdash faire\ (Tourner)}$$
$$\boldsymbol{L(0,2) \wedge F(Est) \wedge \neg S(0,2) \vdash faire(Avancer)}$$
$$\boldsymbol{L(1,2) \wedge F(Est) \wedge \neg S(1,2) \vdash faire\ (Tourner)}$$
$$\boldsymbol{L(1,2) \wedge F(S) \wedge \neg S(1,2) \vdash faire\ (Avancer)}$$
$$\boldsymbol{L(1,1) \wedge F(S) \wedge \neg S(1,1) \vdash faire(Avancer)}$$
$$\boldsymbol{L(1,0) \wedge F(S) \wedge \neg S(1,0) \vdash faire\ (Tourner)}$$
$$\boldsymbol{L(1,0) \wedge F(Est) \wedge \neg S(1,0) \vdash faire(Avancer)}$$
$$\boldsymbol{L(2,0) \wedge F(Est) \wedge \neg S(2,0) \vdash faire\ (Tourner)}$$
$$\boldsymbol{L(2,0) \wedge F(N) \wedge \neg S(2,0) \vdash faire\ (Avancer)}$$
$$\boldsymbol{L(2,1) \wedge F(N) \wedge \neg S(2,1) \vdash faire(Avancer)}$$
$$\boldsymbol{L(2,2) \wedge F(N) \wedge \neg S(2,2) \vdash \cdots}\ \textbf{(Arrêter ou recommencer ?)}$$

Avantages/Inconvénients

L'approche basée sur la logique présente l'avantage d'un fondement théorique solide. Cependant, la complexité des preuves des théorèmes limite l'application de cette approche dans des environnements à forte contraintes temporelles [4]. En outre, cette approche reste inappropriée quand il s'agit des agents ayant des ressources de calcul, de stockage et/ou d'énergie limitées.

Enfin, il faut noter que l'implémentation de ***la fonction See*** effectuant le passage de l'état de l'environnement vers des perceptions représentées via des formules logiques n'est pas une tâche évidente.

1.7.2 Architecture à subsomption

Les architectures de l'IA classique utilisent un découpage vertical de type « détecter- planifier-agir » [5] :

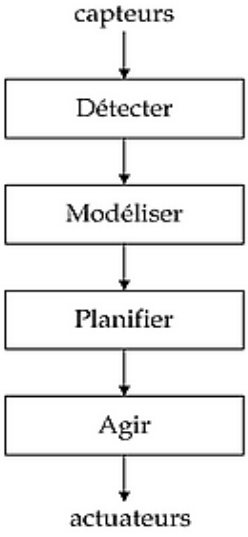


Figure 1.9 Système de contrôle à découpage vertical [5]

Les grandeurs physiques mesurées par les capteurs doivent être reformulées en des informations symboliques exploitables par la couche planification. Cette dernière maintient une représentation symbolique du monde. Les actions décidées sont

passées enfin au module responsable de l'exécution qui est directement connecté au Effecteurs.

En opposition à cette vision, Rodney Brooks (*le père de la robotique*), a développé dans les années 80 ***l'architecture à subsomption*** en utilisant un ***découpage horizontal*** de type « Perception – Action ». A savoir, la subsomption désigne une relation hiérarchique.

Brooks soutient la thèse suivante : « *Le monde est son propre meilleur modèle* ». Il n'est pas nécessaire alors d'utiliser une représentation interne symbolique du monde. Cette architecture de type ***réactive*** repose sur l'idée suivante : « *Le comportement intelligent est produit de l'interaction de l'agent avec son environnement et émerge de l'interaction de plusieurs comportements plus simples* » [3]. L'article « *Elephants don't Play Chess* » [6] décrit les robots développés par Brooks et ses collègues au sein du laboratoire IA au MIT en utilisant l'architecture à subsomption (Robot Allen, Toto, Herbert, Tom and Jerry, ….etc.)

Une architecture à subsomption comporte plusieurs modules (dits **modules de compétence)**, chaque module étant responsable de la réalisation d'une tâche particulière et il est implémenté sous forme de règles : **situation → action**. Les modules sont organisés sous forme hiérarchique.

Modèle formel correspondant [3]

- La fonction **See** a toujours le même rôle sauf que l'entrée des capteurs est supposée très peu transformée.
- Une règle de comportement est une paire (c, a) où $\boldsymbol{c} \subseteq P$ (P est l'ensemble des perceptions) appelé **condition** et **a** est une action.
- Une règle de comportement (c, a) est activée dans un état s SSI $see(s) \in c$.
- Notons par R l'ensemble de toutes les règles de comportement de l'agent ; On définit une relation d'inhibition, $\prec \subseteq RXR$, qui est une relation d'ordre totale *(i.e. relation binaire Transitive, Réflexive et Antisymétrique)*. On écrit a $\prec b$ et on lit a plus prioritaire que b.

Le pseudocode de la fonction Action est comme suit:

```
1.  Fonction action(p: P): A
2.  Var     Règles_activées : ℘(R)
3.  Début
4.  Règles_activées ← {(c,a)|(c,a) ∈ R et p ∈ c}
5.      Pour chaque (c,a) ∈ Règles_activées faire
6.          Si ¬(∃ (c',a') ∈ Règles_activées tel que (c',a') ≺ (c,a)) Alors
7.              Retourner a
8.          Fin-si
9.      Fin-pour
10.     Renvoyer null
11. Fin fonction action
```

Cette fonction détermine tout d'abord l'ensemble des règles de comportements activées. Ensuite, pour chaque règle activée chercher s'il existe une qui soit plus prioritaire. Si aucune n'est trouvée alors **a** est renvoyée. Si aucune règle n'est activée alors renvoyer Null.

Exemple [3]

Imaginons que l'architecture à subsomption est utilisée dans le contrôle d'un robot mobile qui doit faire l'exploration d'une planète à la recherche d'une matière X d'intérêt.

On peut proposer les règles de fonctionnement suivantes pour ce robot:

(R1) Si obstacle détecté alors changer de direction
(R2) Si porter échantillon et se trouver dans la station mère alors déposer échantillon
(R3) Si porter échantillon alors se diriger vers la station
(R4) Si détection de X alors prélever un échantillon
(R5) Si vrai alors explorer aléatoirement (toujours activée)

Avec l'ordre de priorité suivant: $R1 \prec R2 \prec R3 \prec R4 \prec R5$

Avantages/Inconvénients

Par sa simplicité, l'architecture à subsomption est bien adaptée à la résolution de problèmes simples en temps réel (comme la navigation réactive d'un robot mobile). L'insertion de nouveaux comportements se fait facilement sans avoir à réécrire ceux déjà existants [5].

Néanmoins, en tant qu'architecture réactive, il sera impossible de doter l'agent de la faculté de raisonnement ou d'apprentissage (aucun état interne n'est maintenu et aussi aucune fonction d'utilité n'est définie). En outre, il n'existe pas une méthodologie commune pour la construction des agents selon cette architecture : ce n'est que par expérimentation qu'on vérifie la validité de notre conceptualisation [7].

1.7.3 Architecture BDI (Belief, Desire, Intention)

Cette architecture a été proposée par Michael Bartman en 1987, elle se base sur la compréhension du raisonnement pratique: processus de décider, étape par étape, quelle action exécuter pour aboutir à un objectif [3]. Dans ce processus, il faut d'abord fixer les buts à atteindre (délibération) c'est-à-dire se demander quoi faire ? Puis déterminer comment faire pour les atteindre ? (articulation moyens/fins) [4].

1.7.3.1 Description informelle du modèle

Le modèle BDI est basé sur trois attitudes mentales, à savoir: les croyances, les désirs et les intentions.

Croyances : correspondent à la représentation de l'agent de l'état de son environnement. Les croyances ne sont pas forcément vraies.

> **Exemple:** un étudiant croit qu'il y aura des cours *Dimanche* prochain ; Alors que c'est un jour férié par exemple.

Désirs : correspondent aux options disponibles à l'agent, les différents désirs peuvent être en conflit.

> **Exemple :** l'étudient peut assister le TD demain à 8h mais il peut aussi aller voir son médecin à 8h.

Intentions : réfèrent à l'engagement de l'agent à la réalisation des désirs mais aussi au plan lui permettant de les achever. Dans le modèle BDI, le raisonnement est

restreint à la réalisation des intentions. Bien sûr toute option en conflit avec les intentions de l'agent doit être systématiquement annulée.

> **Exemple :** l'étudiant décide d'assister au TD demain à 8H. Maintenant l'étudiant doit raisonner sur comment arriver à l'heure. Le matin à 7:50 H l'étudiant rencontre un collègue qui lui propose d'aller réviser en salle de lecture : cette proposition est directement refusée !

En résumé : étant données des croyances sur l'environnement, l'agent peut avoir plusieurs désirs mais celles choisies (engagements) deviennent les intentions de l'agent ; les intentions déterminent les actions à exécuter (le plan).

Toutefois, un agent doit reconsidérer ou réviser ses intentions afin de supprimer celles déjà réalisées, celles devenues impossibles ou celles dont la motivation n'est plus présente. Certes, cette opération est coûteuse en termes des ressources computationnelles.

Un dilemme se pose ici : d'une part, un agent doit reconsidérer ses intentions (pour les raisons déjà mentionnées), d'autre part, un agent qui reconsidère constamment ses intentions ne consacre pas suffisamment de temps dans leurs réalisations et risque de ne les jamais accomplir. Il faut alors trouver un compromis : déterminer la fréquence de reconsidération des intentions selon la dynamique de l'environnement.

1.7.3.2 Spécification formelle d'un agent BDI

L'état d'un agent BDI, à tout moment, est défini par le triplet (B,D,I) tel que: $\boldsymbol{B} \subseteq \boldsymbol{Bel}, \boldsymbol{D} \subseteq \boldsymbol{Des}, \boldsymbol{I} \subseteq \boldsymbol{Int}$, et :

- Bel: l'ensemble de toutes les croyances possibles sur l'état de l'environnement.
- Des : l'ensemble de tous les désirs possibles de l'agent.
- Int : l'ensemble de toutes les Intentions possibles de l'agent.

Le comportement d'un agent BDI est alors défini par [3] :

- La fonction de révision de ses croyances qui calcule ses nouvelles croyances à partir des croyances courantes et de nouvelles perceptions de son environnement :

$$\boldsymbol{brf}: \wp(\boldsymbol{Bel}) \times \boldsymbol{P} \rightarrow \wp(\boldsymbol{Bel})$$

- Une fonction **options** de génération de ses options qui représentent ses désirs ou choix possibles conformément à ses intentions ; basée sur les croyances qu'il a de son environnement et sur ses intentions :

$$\boldsymbol{options}: \wp(\boldsymbol{Bel}) \times \wp(\boldsymbol{Int}) \rightarrow \wp(\boldsymbol{Des})$$

- Fonction filtre : met à jour les intentions de l'agent (par exemple faire exclure des intentions irréalisables, garder celles non encore réalisées, exploiter de nouvelles opportunités …)

$$\boldsymbol{filter}: \wp(\boldsymbol{Bel}) \times \wp(\boldsymbol{Des}) \times \wp(\boldsymbol{Int}) \rightarrow \wp(\boldsymbol{Int})$$

Notons que filtrer(B,D,I) est inclus dans I union D : les nouvelles intentions sont soit les anciennes intentions ou de nouvelles options adoptées.

- Fonction exécuter : retrourne une action exécutable

$$\boldsymbol{execute}: \wp(\boldsymbol{Int}) \rightarrow \boldsymbol{A}$$

Le pseudo-code de la fonction action de l'agent BDI correspond à ce qui suit:

1. Fonction action (p :P) : A
2. Début
3. $B \leftarrow brf(B,p)$
4. $D \leftarrow options\ (B,I)$
5. $I \leftarrow filter(B,D,I)$
6. Retourner $execute\ (I)$
7. Fin

Enfin, le model BDI est très intuitif avec une décomposition fonctionnelle claire. Ce modèle a été implémenté avec succès notamment dans le système PRS (*Procedural Reasoning System*) et son successeur dMars (*distributed Multi-Agent Reasonning System*).

1.7.4 Architectures hybrides

Un agent réactif a l'avantage d'une réponse à temps du moment qu'il fonctionne avec des règles simples de type : stimulus – action. Un tel agent ne maintient aucune représentation de son environnement. Inversement, un agent proactif ou délibératif maintient une représentation symbolique de son environnement, manipule cette représentation afin de générer un comportement plus ou moins sophistiqué. L'inconvénient d'un agent *qui raisonne* est qu'il est inadapté aux environnements dynamiques.

Dans un problème issu du monde réel (environnement hautement dynamique), on a besoin de la réactivité pour s'adapter aux changements survenant dans l'environnement et de la délibération afin de réaliser des tâches plus ou moins complexes. C'est cette constatation qui est derrière la proposition des architectures hybrides qui combinent comportement réactif et proactif au sein d'un même agent.

Selon la propagation du flux données/contrôle, on distingue des architectures verticales ou horizontales comme le montre la figure ci-dessous.

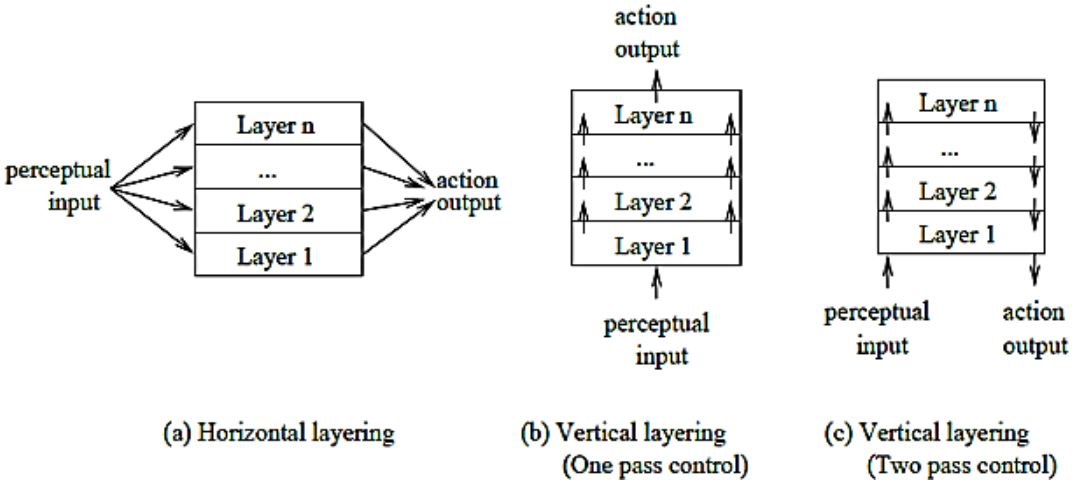


Figure 1.10 Flux de données/contrôle dans les trois architectures en couches [3].

Dans une architecture horizontale, chaque couche propose une action et on aura alors besoin de sélectionner une seule. Dans une architecture verticale, un seul niveau se charge de la prise de décision. Si la couche inférieure est incapable de décider alors le contrôle est passé à la couche supérieure et ainsi de suite.

NB. Quand il y a deux passes : on parle d'activation Bottom-up et d'exécution Top-down.

Prenons comme exemple l'architecture *Interrap* de *Jörg Müller (Integration of Reactive Behavior and Rational Planning = Intégration du comportement réactif et planification rationnelle)* [8] qui génère comportement réactif, proactif et social. Il s'agit d'une architecture verticale à deux passes [4]. Cette architecture a été simulée et testée pour un problème de transport de marchandises par des robots mobiles (il s'agit d'un ***SMA coopératif***).

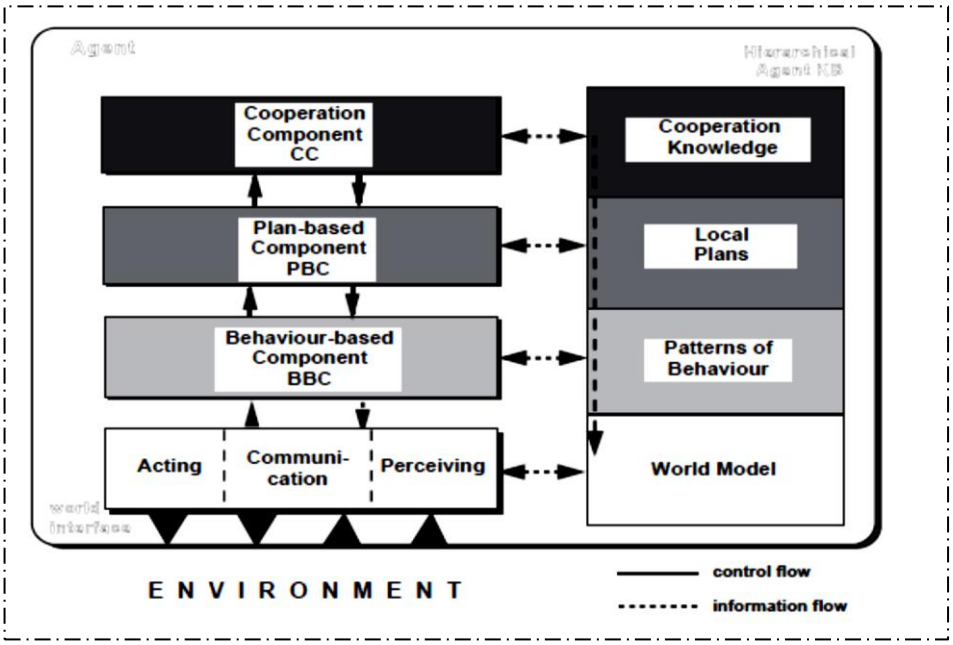


Figure 1.11 Model d'un agent INTERRAP [8]

Sur la figure 1.11 (BC pour base de connaissances) :

- BC-Monde (*World Model*) représente les croyances sur l'environnement.
- BC-Planification (*Patterns of Behaviour & Local Plans*) correspond à une bibliothèque de plans.
- BC-Sociale (*Cooperation knowledge*) représente les croyances de l'agent sur les autres agents du système et notamment sur leurs capacités de lui aider à atteindre ses buts.

Les buts d'un agent InteRRaP sont divisés en trois catégories:

- Réactions (assurées par BBC): ce sont des buts simples à accomplir en fonction des perceptions sur l'environnement.
- Buts locaux (assurés par PBC) : ce sont des buts que l'agent peut accomplir lui-même.
- Buts coopératifs (assurés par CC) : ce sont les buts qui peuvent être accomplis uniquement par une coopération avec d'autres agents dans le système.

Exemple

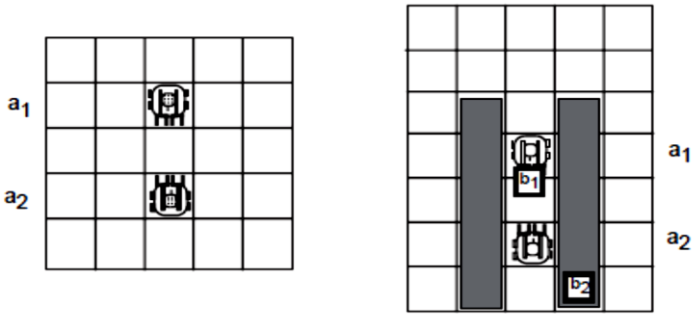


Figure 1.12 Situations de conflit entre deux robots Interrap [8].

Pour la première scène (à gauche), les deux robots peuvent faire des simples mouvements pour éviter la collision (reculer, tourner, etc.). Donc, le problème est réglé au niveau de la couche responsable du *comportement réactif*. Pour la deuxième scène (à droite), le robot a1 ne peut pas déposer sa charge puisque le robot a2 (qui vient de déposer la sienne) le bloque. Une solution intelligente (avec un minimum de déplacements pour les deux agents) consiste à déléguer le dépôt de la charge de a1 à a2. La négociation d'un tel plan conjoint est assurée par la couche responsable du *comportement social*.

1.8 Conclusion

Ce chapitre donne une vue d'ensemble sur la conception des agents intelligents. En parallèle, l'attention des étudiants est orientée vers le domaine pluridisciplinaire de la robotique mobile. Les connaissances théoriques exposées dans ce chapitre peuvent être renforcées, par exemple, par l'apprentissage du langage de programmation des systèmes multi-agent: «*AgentSpeak*» [9].

Dans le chapitre suivant, nous nous intéressons à la représentation des connaissances via les ontologies. Nous parlerons, entre autres, sur leur apport à la communication dans les systèmes multi-agent.

Références du chapitre

[1] S. Russell and P. Norvig, *Intelligence artificielle*. Paris: Pearson Education, 2010.

[2] J. Ferber, *Les systèmes multi-agents. Vers une intelligence collective*. Paris : InterEditions, 1995.

[3] G. Weiss, *Multiagent systems: a modern approach to distributed artificial intelligence*. Cambridge, Mass. [u.a.]: MIT Press, 2006.

[4] L. Frécon and O. Kazar, *Manuel d'intelligence artificielle*. Lausanne: Presses polytechniques et universitaires romandes, 2009.

[5] P. Arnaud, *Des moutons et des robots*. Lausanne: Presses polytechniques et universitaires romandes, 2000.

[6] R. Brooks, *Elephants don't play chess*, Robotics and Autonomous Systems, vol. 6, no. 1-2, pp. 3-15, 1990.

[7] M. Luck, R. Ashri and M. D'Inverno, *Agent-based software development*. Boston: Artech House, 2004.

[8] J.P. Müller and P. Markus, *The agent architecture InteRRaP : concept and application*. Rapport de recherche, DFKI Saarbrücken (Allemagne), 1993.

[9] J. Hübner and M. Wooldridge, *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2008.

CHAPITRE 2

Connaissance et Ontologie

2.1 Introduction

L'un des problèmes fondamentaux en Intelligence Artificielle est la représentation symbolique des connaissances : sous forme de formules logiques (*cas d'un système formel*) ou de règles de production (*cas d'un système expert*), etc. Le niveau symbolique a été considéré pour longtemps comme le niveau adéquat pour la modélisation de l'activité mentale [1]. Insatisfait de cette vision, ***Allen Newell*** [2] propose d'ajouter un niveau d'abstraction dit « niveau de connaissance » au-dessus du « niveau symbolique ». À ce niveau doit être définie la connaissance indépendamment de sa représentation symbolique.

Le premier objectif de ce chapitre, intitulé « Connaissance et Ontologie », est de montrer l'apport des ontologies au domaine de la représentation de connaissance. Initialement, est d'une manière très simplifiée, on peut dire qu'une ontologie est le support de tout type de connaissance : classes d'objets, propriétés et relations entre objets mais aussi faits, règles et contraintes. A savoir, la conceptualisation d'une ontologie doit être spécifiée au niveau cognitif. Mais, le présent cours va au-delà de la phase de conceptualisation : nous abordons également la formalisation des ontologies en logique de description et leur encodage en langage OWL (*Ontologie Web Language*).

2.2 Niveau de connaissance

Une connaissance ne contient pas dans sa ***forme*** ce qui fait d'elle une connaissance [1]. Par forme, on désigne la représentation de la connaissance sur ordinateur en termes de structures de données avec les opérations de manipulation et les procédures d'accès. Les connaissances se rapportent plutôt au **contenu conceptuel** des structures de données [3].

D'après Newell, la description d'un processus de résolution de problèmes en termes de connaissances nécessite l'introduction d'un nouveau niveau de description indépendant de toute représentation informatique : le niveau cognitif (***Knowledge Level***). Newell a proposé de continuer le parallèle entre le comportement de l'homme et celui de la machine comme l'illustre la figure ci-dessous.

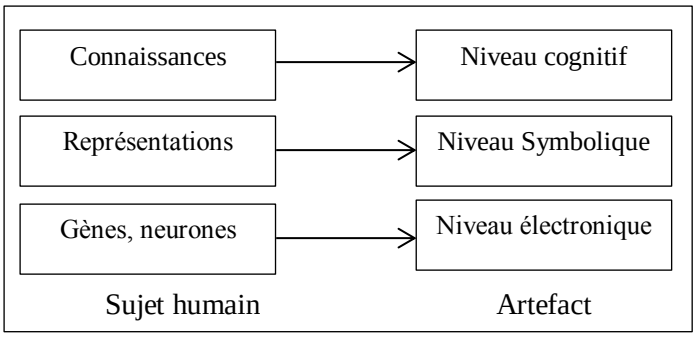


Figure 2.1 Les niveaux de description de comportement selon Newell [1]

Le niveau physique de l'artefact correspond à tout ce qui est diode, transistor, circuit électronique, etc. Le niveau symbolique quant à lui inclut tout ce qui est langage de programmation.

Le niveau cognitif doit permettre de décrire les intentions et les justifications rationnelles du comportement d'un système. A ce niveau, la connaissance prend la forme **de buts** à atteindre, de **savoir** sur le monde et d'**actions** sur l'environnement. Ces trois éléments interagissent selon le principe ***de rationalité.*** Ce principe consiste à choisir, en fonction du savoir sur le monde, les actions susceptibles de conduire aux buts visés [1].

Exemple

Donner une spécification au niveau cognitif du processus de classification.

Pour répondre à cette question, il faut identifier les connaissances nécessaires au traitement du problème de classification séparément de toute implémentation opérationnelle [1] :

- ***But :*** dans un problème de classification, l'objectif est d'identifier la classe d'appartenance d'un objet.
- ***Savoir*** : observations enregistrées sur l'objet en question ainsi que les classes avec leurs caractéristiques.
- ***Actions :*** re-description des observations en termes des caractéristiques, comparaisons, un éventuel affinement de la classification obtenue.

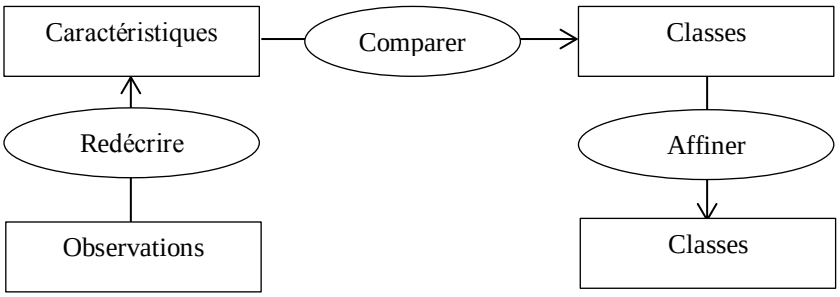


Figure 2.2 Description du processus de classification au niveau cognitif [1].

Pour plus d'éclaircissement, considérons un problème très simplifié: classification d'une maison dans une des deux classes : «*bonne maison*» et « *maison moins bonne*». Une *bonne maison* possède trois caractéristiques : elle est grande, avec jardin et bien située. Une maison est *moins bonne* si une des caractéristiques précédentes n'est pas vérifiée.

Etant donnée une maison, on a les observations suivantes : superficie (X), avec jardin ou non (Y) et distance par rapport au centre-ville (Z).

Re-description (S1 et S2 sont des seuils):

Si $X \geq S_1$ alors grande

Si $Z \leq S_2$ alors bien située (puisque proche du centre-ville)

Dans l'affinement, on peut proposer une nouvelle classe de maisons : celles qui ne vérifient aucune des caractéristiques d'une bonne maison.

2.3 Ontologies

Une **donnée** s'élabore à partir d'observation elle est simple et brute et n'a pas d'utilité immédiate (*par exemple 40*). Cependant, l'**information** est une donnée explicite et signifiante (*par exemple: 40°C : maintenant on comprend qu'il s'agit d'une Température*). La **connaissance** est une information contextualisée à laquelle on peut attribuer une valeur de vérité [4] (*par exemple: le malade de la chambre 3 du service pédiatrie a une température de 40°C*).

L'**ontologie** est un support de ***connaissance*** utilisé par la communauté IA depuis les années 80. Mais que désigne-t-on par Ontologie ? Et d'où vient ce terme ?

2.3.1 Origine du terme Ontologie

Les philosophes depuis Aristote se sont intéressés à ce qui existe et comment le décrire mais aussi comment le placer parmi ce qui existe déjà. L'Ontologie, en philosophie, est une branche de la Métaphysique qui s'intéresse à l'existence. En effet, ce terme est construit à partir des racines grecques ***ontos*** qui veut dire ce qui existe, l'être ou l'existant, et ***logos*** qui veut dire étude : d'où l'interprétation « *étude de l'être et par extension de l'existence* ».

2.3.2 Définition de l'ontologie en Informatique

Les chercheurs en IA ont adopté les ontologies comme modèle computationnel qui permet un certain raisonnement automatique. Pour certains chercheurs, les ontologies computationnelles présentent une sorte de philosophie appliquée. La

définition de l'ontologie la plus répandue, en informatique, est celle donnée par Gruber [5] : ***« Une ontologie est une spécification explicite et formelle d'une conceptualisation partagée ».***

- Spécification explicite : les concepts, propriétés, relations, contraintes et les axiomes d'un domaine de connaissance sont définis d'une manière claire et précise.
- Formelle: interprétable par une machine ce qui permet de mener des raisonnements et déduire des nouvelles connaissances.
- Conceptualisation : modèle abstrait d'un phénomène dans le monde.
- Partagée : il s'agit de connaissances communes (sens & vocabulaire). Une ontologie est le produit d'un consensus au sein d'une communauté.

Les ontologies sont utilisées comme *référentiels de communication* entre humains et entre agents artificiels. Elles réduisent les types de conflits suivants:

- **Conflit conceptuel :** un même terme est interprété différemment.
  Par exemple : les chercheurs en IA n'ont pas une vision commune sur ce qui est l'intelligence. Chacun l'interprète d'un certain point de vue.
- **Conflit terminologique :** on est d'accord pour le sens d'un concept mais pas pour le nom donné au concept.
  Par exemple : doit-on utiliser Taxonomie ou Taxinomie pour désigner une classification?

Pour mieux comprendre ce qui est une ontologie, nous positionnons cette dernière par rapport à des notions très proches, à savoir*: « Vocabulaire »* et *« Base de connaissance ».*

2.3.2.1 Une Ontologie est un « Vocabulaire très riche »

Selon leurs consistances, on distingue différents types de vocabulaires:

- Vocabulaire contrôlé: il s'agit d'une ***liste de termes*** liés à un domaine d'intérêt.
- Taxonomie: ***liste de termes*** avec une relation de subsomption entre les termes.
- Thésaurus: organisation hiérarchique de ***termes*** liés entre eux par des relations de "synonymie" et de "terme associé".
- Ontologie : vocabulaire plus consistant, il peut être vu comme un réseau *(différents types de relations)* de ***concepts*** *(au fait, un concept est plus qu'un simple terme).*

2.3.2.2 Ontologie versus Base de connaissance

L'ontologie capture et représente la connaissance d'un domaine sans se rattacher à une application particulière *(dans ce cas, on parle d'ontologie de domaine)*. Elle reflète une compréhension commune du domaine et elle est de ce fait **réutilisable**. En revanche, une base de connaissance est construite dans l'esprit de résolution d'un problème lié à un domaine donné. De ce fait, elle doit contenir de la connaissance opérationnelle et seulement la partie de la connaissance conceptuelle nécessaire à la résolution du problème posé [6].

De ce point de vue, les ontologies peuvent être utilisées comme un ***méta-modèle*** pour la construction des systèmes à base de connaissance.

2.4 Cycle de vie des ontologies

Le cycle de vie d'une ontologie, inspiré du génie logiciel, peut être schématisé comme dans la figure ci-dessous :

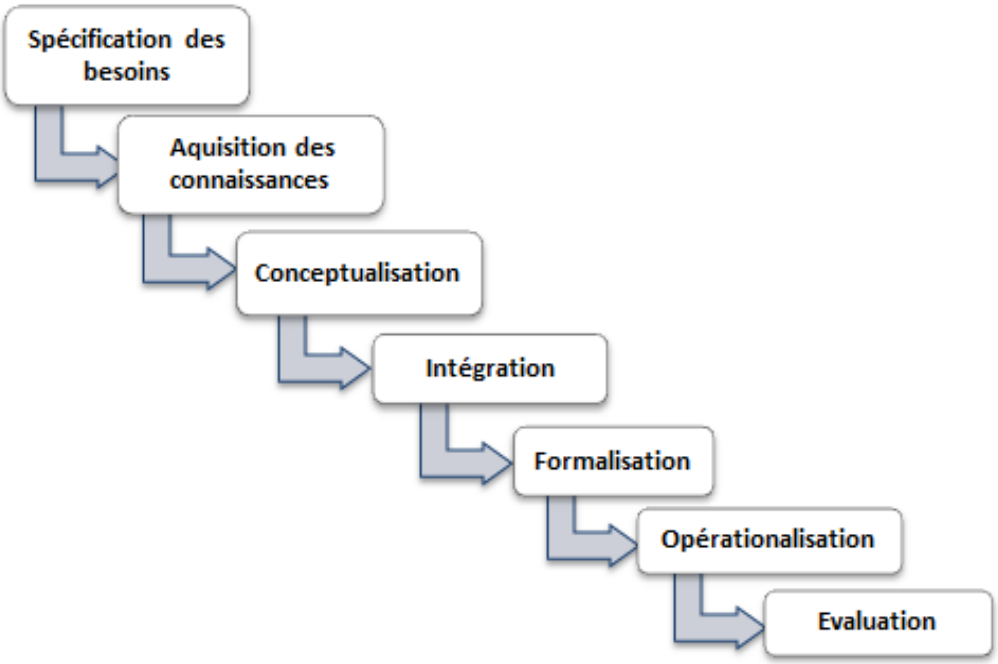


Figure 2.3 Cycle de vie d'une ontologie.

- **Spécification**: description de la portée de l'ontologie, ses utilisateurs ainsi que le niveau de formalisation souhaité.
- **Acquisition des connaissances**: identifier les termes de l'ontologie et leur définition et cela via l'analyse d'un corpus documentaire et/ou par interview des spécialistes/experts.
- **Conceptualisation**: l'ingénieur de connaissance identifie les concepts, les relations, les axiomes,… etc.
- **Intégration**: recherche d'ontologies existantes qui peuvent être réutilisées.
- **Formalisation:** formaliser le modèle conceptuel obtenu *(dans notre cours, en logique de description)*.
- **Opérationnalisation**: codage de l'ontologie dans un langage opérationnel *(dans notre cours, en langage OWL).*
- **Évaluation**: vérifier et valider l'ontologie *(de point de vue cohérence, complétude, etc.)*

Remarque : Bien sûr ce processus n'est pas linéaire on peut réviser notre ontologie voire même mettre à jour nos besoins.

2.5 Phase de Conceptualisation

Une alternative possible dans la *Phase de conceptualisation* consiste à suivre les étapes de la méthode **METHONTOLOGY** [7] considérée parmi les méthodes les plus complètes pour la construction d'une ontologie :

1. Construire un glossaire de termes.

2. Construire les arbres de classification des concepts (les hiérarchies).
3. Tracer le diagramme des relations binaires entre concepts.
4. Pour chaque arbre de classification construire :
    1. une table de concepts
    2. une table de relations binaires
    3. une table des attributs
    4. une table des axiomes
    5. une table des instances
    6. une table des assertions

Remarque

Les axiomes expriment la sémantique du domaine. Ils déterminent comment les relations/concepts peuvent être utilisés d'une manière opérationnelle (aller au-delà de la terminologie). Exemples d'axiomes dans une ontologie:

- ✓ Propriétés algébriques d'une relation (symétrie, réflexivité, transitivité)
- ✓ La subsomption entre concepts et/ou relation
- ✓ Les cardinalités d'une relation
- ✓ L'incompatibilité entre concepts (i.e. la disjonction)

Exemple (*Il s'agit d'une initiation à la conceptualisation, voir le TD pour approfondir*)

« Un robot mobile est un agent intelligent. un agent intelligent est capable d'exécuter une tâche d'une manière autonome, il est capable de percevoir l'état de son environnement et de se déplacer dans celui-ci ; un robot mobile possède une architecture … une architecture de robotique peut être réactive, proactive ou hybride…iRobot Roomba 620 est un robot aspirateur il a les caractéristiques suivantes [1]: Diamètre du robot (cm) : 34, Poids du robot (kg) : 3,6, Puissance (W) : 33, Capacité du réservoir (l) : 0,498.Imaginons qu'un certain client (c1) utilise son nouvel aspirateur iRobot Roomba 620 (a1) pour nettoyer sa maison (m1) »

A partir de ce texte, on peut dégager :

1. Les concepts : robot mobile, agent intelligent, environnement, architecture de robotique, robot aspirateur…
2. Deux hiérarchies (y compris les relations de subsomption est-un):
   H1 : Agent intelligent
   –Robot mobile
   – Robot aspirateur
   – iRobot Roomba 620
   H2: Architecture de robotique
   – Architecture réactive
   – Architecture proactive
   – Architecture hybride
3. Les relations binaires: posséder, se déplacer dans, nettoyer, …
4. Les attributs du concept robot aspirateur : diamètre, poids, puissance, capacité du réservoir,…
5. Les instances: a1, *c1, m1,…*
6. Les assertions : « a1 (nettoyer) m1 », « a1 (se déplacer dans) m1 »,…

[1] *https://www.amazon.fr/iRobot-Roomba-620-Aspirateur-Autonome/dp/B008ZAYZLM*

2.6 Logiques de Description

Plus récemment, les langages de représentation des ontologies à base des logiques de description sont devenus très populaires. Ces langages sont apparus notamment dans le contexte du Web sémantique [8]. Les logiques de description sont des fragments suffisamment expressifs de la logique des prédicats[2] et qui sont souvent décidables.

2.6.1 Syntaxe et Sémantique des LDs

La **sémantique d'une LD** est donnée au moyen d'un domaine d'interprétation Δ^I qui est un ensemble non vide d'individus et d'une fonction d'interprétation **I** qui fait correspondre à un concept un sous-ensemble de Δ^I et à un rôle un sous-ensemble de Δ^I X Δ^I . Le tableau ci-dessous récapitule la Syntaxe et la sémantique des Logiques de Description [9].

Logique	Constructeur	Syntaxe	Sémantique
$\mathcal{AL}$	Concept atomique	A	$A^I \subseteq \Delta^I$
	Rôle atomique	R	$R^I \subseteq \Delta^I \times \Delta^I$
	Concept le plus générale	T	Δ^I
	Concept le plus spécifique	$\bot$	$\emptyset$
	Négation atomique	$\neg A$	$\Delta^I \backslash A^I$
	Conjonction	$C \cap D$	$C^I \cap D^I$
	Quantification universelle	$\forall R.C$	$\{a \in \Delta^I \mid \forall b(a,b) \in R^I \rightarrow b \in C^I\}$
	Quantification existentielle non qualifiée	$\exists R.T$	$\{a \in \Delta^I \mid \exists b(a,b) \in R^I\}$
U	Disjonction ou union de deux concepts	$C \cup D$	$C^I \cup D^I$
$\mathcal{E}$	Quantification existentielle	$\exists R.C$	$\{a \in \Delta^I \mid \exists b(a,b) \in R^I \wedge b \in C^I\}$
$\aleph$	Restriction de nombre	$\leq nR$ $\geq nR$ $= nR$	$\{a \in \Delta^I \mid \lvert\{b \mid (a,b) \in R^I\}\rvert \geq n\}$ $\{a \in \Delta^I \mid \lvert\{b \mid (a,b) \in R^I\}\rvert \leq n\}$ $\{a \in \Delta^I \mid \lvert\{b \mid (a,b) \in R^I\}\rvert = n\}$
F	Rôle fonctionnel	$\leq 1\ R$	$\{a \in \Delta^I \mid \lvert\{b \mid (a,b) \in R^I\}\rvert \leq 1\}$
C	Négation de concepts arbitraire	$\neg C$	$\Delta^I \backslash C^I$
Q	Restriction de nombre qualifiée	$\leq nR.C$ $\geq nR.C$ $= nR.C$	$\{a \in \Delta^I \mid \lvert\{b \in \Delta^I \mid (a,b) \in R^I \wedge b \in C^I\}\rvert \leq n\}$ $\{a \in \Delta^I \mid \lvert\{b \in \Delta^I \mid (a,b) \in R^I \wedge b \in C^I\}\rvert \geq n\}$ $\{a \in \Delta^I \mid \lvert\{b \in \Delta^I \mid (a,b) \in R^I \wedge b \in C^I\}\rvert = n\}$
S	Clôture transitive de rôle	R^+	$\cup_{i \geq 1} (R^I)^i$
H	Rôle hiérarchique	$R \subseteq S$	$R^i \subseteq S^i$
I	Rôle inverse	R^-	$\{(b,a) \in \Delta^I \times \Delta^I \mid (a,b) \in R^I\}$
O	Description des concepts par énumération d'individus nommés	$\{a_1\} \cup \ldots \{a_n\}$	$\{a_1{}^I\} \cup \ldots \cup \{a_n{}^I\}$

Tableau 2.1 Logiques de description.

[2] La logique des prédicats est semi-décidable.

En augmentant la logique de base $\boldsymbol{AL}$ avec les différents constructeurs on obtient une famille de logiques de description, en l'occurrence :

- $\boldsymbol{AL}\mathcal{E}\ \boldsymbol{N}$ c'est la logique AL à laquelle sont ajoutées la quantification existentielle et la restriction de nombre non qualifiée.
- Logique **ALC** est équivalente à **ALU**$\mathcal{E}$ puisque grâce à la négation non atomique on peut définir U et $\mathcal{E}$:

  $\neg(C \cap D) \equiv \neg C \cup \neg D$

  $\neg\forall R.C \equiv \exists R.\neg C$
- La logique ALC augmentée par les constructeurs H et S donne la logique **SH**.
- SH étendue avec I et F donne la logique **SHIF** (base du langage OWL-Lite).
- SHIF est clairement la logique SH + I + F.
- **SHOIN** est la logique SH + O+ I + N (qui est à la base du langage OWL-DL).

Remarques

$\neg(\geq n\ R) \equiv \leq (n-1)R$

$\neg(\leq n\ R) \equiv \geq (n+1)R$

$\neg\neg\ C \equiv C$

Une ***base de connaissance*** $\boldsymbol{\Sigma}$ écrite en LD contient deux composantes [9]:

- **T (TBox) :** représente la **t**erminologie d'un domaine sous forme de définitions de concepts et de rôles. **T** est un ensemble fini d'axiomes terminologiques :

Axiome	Lecture
$\boldsymbol{C \equiv D}$	Le Concept C est par définition égal au concept D (*on utilise la même syntaxe pour définir l'équivalence entre rôles*)
$\boldsymbol{C \subseteq D}$	Le Concept C est par définition subsumé par le concept D (*on utilise la même syntaxe pour exprimer la subsomption entre rôles*)

- **A (ABox)** : contient les ***a**ssertions* sur les individus du domaine considéré en utilisant les termes de l'ontologie. **A** est un ensemble fini d'assertions:

Assertion	Lecture
$\boldsymbol{C}\ (\boldsymbol{a})$	a est un individu du concept C
$\mathbf{R}\ (\boldsymbol{b}, \boldsymbol{c})$	Les individus b et c sont liés par R

Exemple (Syntaxe)

Considérons les deux concepts atomiques ***Personne*** et ***Féminin*** et le rôle atomique ***aEnfant***. On peut alors définir les concepts suivants:

- Homme : Personne ⊓**¬Féminin**
- Personnes avec au moins un enfant : Personne⊓∃**aEnfant**
- Personnes dont tous les enfants sont des garçons : Personne⊓∀aEnfant**. ¬Féminin**

Exemple inspiré de la référence [10] (Sémantique)

Soit la base de connaissance qui contient les axiomes suivants :

$Professeur \subseteq MembreFaculté$

$Professeur(Ali)$

$Enseigne\_à(Ali, univJijel)$

L'interprétation suivante est-elle un modèle pour cette BC ?

$\Delta^I = \{a, b, c\}$

$(Ali)^I = a$

$(univJijel)^I = b$

$(Professeur)^I = \{c\}$
$(MembreFaculté)^I = \{a, c\}$
$(Enseigne\_à)^I = \{(c, b)\}$

Réponse : Non, pour les raisons suivantes :

- $(Ali)^I$ n'est pas inclut dans $(Professeur)^I$ cependant d'après la BC Ali est un professeur.
- D'après la BC $((Ali)^I, (univJijel)^I)$ doit appartenir à $(Enseigne\_à)^I$ alors que ce n'est pas le cas.

2.6.2 Propriétés [9]

- Satisfiabilité (consistance): Un concept C est satisfaisable, pour une interprétation I, SSI $C^I \neq \phi$.
  Exemple le concept $Homme \cap \neg Homme$ est insatisifable.
- Subsomption: Un concept C est subsumé par un concept D, pour une interprétation I, si et seulement si $C^I \subseteq D^I$
  Exemple $Père \subseteq Homme$
- Equivalence : Un concept C est équivalent à un concept D, pour une interprétation I, si et seulement si $C^I = D^I$
- Incompatibilité ou disjonction: Deux concepts C et D sont incompatibles, pour une interprétation I, si et seulement si $C^I \cap D^I = \phi$
  Exemple : GrandPère ∩ GrandeMère⊆⊥

Exemple (Exercice Résolu du sujet d'examen 2015/2016)

Imaginons qu'on dispose de deux ontologies décrivant : « les Enseignants, les Chercheurs ainsi que les Documents rédigés par ces derniers» :

- La première ontologie est construite selon le point de vue « tâche pédagogique» (utilisée par exemple au sein du département) ; Voici son ABOX complète (Tous les individus sont mentionnés)

ABOX de la première ontologie		
Enseignant (Omar)	Enseignant (Ali)	Enseignant (Sonia)
Enseignant (Lila)	Enseignant (Dina)	
Enseignant Vacataire ()	Chargé de cours (Ali)	Chargé de TD (Lila)
Chargé de TP (Omar)	Chargé de TP (Sonia)	Chargé de TP (Dina)
Support de cours (doc3)	Support de cours (doc4)	

- La deuxième ontologie est construite selon le point de vue « grade scientifique» (utilisée par exemple au sein du laboratoire de recherche) ; Voici son ABOX complète:

ABOX de la deuxième ontologie	
Maitre-assistant classe A (Omar)	Maitre-assistant classe A (Sonia)
Maitre de conférences (Lila)	Professeur (Ali)
Enseignant-chercheur (Omar)	Enseignant-chercheur (Ali)
Enseignant-chercheur (Sonia)	Enseignant-chercheur (Lila)
Article de recherche (doc1)	Article de recherche (doc2)

On veut fusionner les deux ontologies. Exprimer en LD la TBOX de l'ontologie fusionnée (résultante) tout en donnant les justifications formelles adéquates.

Réponse : Clairement, le passage à la TBOX de l'ontologie fusionnée se fera à base des interprétations des concepts données dans les ABOX :

TBOX en LD	Justification formelle
Chargé de cours ≡professeur Chargé de TD ≡Maitre de conf…	Un concept C est **équivalent** à un concept D si et seulement si $C^I = D^I$
Enseignant-chercheur ⊑Enseignant MAA⊑ Chargé de TP…	Un concept C est **subsumé par** un concept D si et seulement si $C^I \subseteq D^I$
Support de cours ⊑ ***T*** Article de recherche ⊑ ***T*** Enseignant ⊑ ***T*** ….	Ne sont subsumés par aucun concept défini dans les deux ontologies
Enseignant Vacataire ⊑ ⊥	Un concept est non satisfiable alors il est subsumé par ⊥ ; Sachant qu'un concept C est satisfaisable pour une interprétation I SSI $C^I \neq \phi$.

2.6.3 Problèmes d'inférences

Etant donnée une BC en LD, les problèmes d'inférence que nous souhaitons résoudre sont les suivants [10] (on parle de concepts ou de classes indifféremment):

1) Satisfiabilité de la BC dans l'ensemble (possède-t-elle au moins un modèle ?)
2) Satisfiabilité de classes.
3) Inclusion de classes.
4) Equivalence de classes.
5) Disjonction de classes.
6) Appartenance d'un individu à une classe.
7) Recherche de tous les individus connus d'une classe.

Notons bien que les problèmes de 2 à 7 se réduisent au problème 1 qui peut être résolu en utilisant l'algorithme des ***Tableaux Sémantiques***. Cet algorithme est basé sur l'idée de ***preuve par réfutation*** : c'est-à-dire afin de prouver qu'une formule est satisfiable on montre que sa négation est une contradiction.

2.6.3.1 Réduction au non satisfiabilité

Soit K une base de connaissance en LD, la réduction au non satisfiabilité se fait ainsi [10] :

- **Non satisfiabilité d'une classe**

$(K \vDash C \subseteq \perp)$ SSI $K \cup \{C(a)\}$ est non satisfiable tel que a est un nouvel individu (ne figure pas déjà dans K).

- **Subsomption**

$(K \vDash C \subseteq D)$ SSI $K \cup \{(C \cap \neg D)(a)\}$ est non satisfiable tel que a est un nouvel individu (ne figure pas déjà dans K).

Soulignons que $C \subseteq D \equiv \neg C \cup D$

- **Equivalence de classes**

$(K \vDash C \equiv D)$ SSI $K \vDash C \subseteq D$ et $K \vDash D \subseteq C$

- **Disjonction de classes**
  $(K \vDash C \cap D \subseteq \bot)$ SSI $K \cup \{(C \cap D)(a)\}$ est non satisfiable tel que a est un nouvel individu.
- **Appartenance d'un individu à une classe**
  $(K \vDash C(a))$ SSI $K \cup \{\neg C(a)\}$ est non satisfiable.
- **Recherche de tous les individus connus d'une classe**
  Il faut vérifier pour tous les individus a, si $K \vDash C(\boldsymbol{a})$.

2.6.3.2 Algorithme des Tableaux Sémantiques pour la logique ALC

Avant d'appliquer l'algorithme, il faut réécrire la BC en utilisant seulement les symboles de l'union et de l'intersection. Ensuite, il faut transformer la BC à son équivalent en NNF (*Negation Normal Form*) comme suit [10]:

$NNF(C) = C \; si \; C \; est \; un \; concept \; atomique$
$NNF(\neg C) = \neg C \; si \; C \; est \; un \; concept \; atomique$
$NNF(\neg\neg C) = NNF(C)$
$NNF(C \cup D) = NNF(C) \cup NNF(D)$
$NNF(C \cap D) = NNF(C) \cap NNF(D)$
$NNF(\neg(C \cup D)) = NNF(\neg C) \cap NNF(\neg D)$
$NNF(\neg(C \cap D)) = NNF(\neg C) \cup NNF(\neg D)$
$NNF(\forall R.C) = \forall R.NNF(C)$
$NNF(\exists R.C) = \exists R.NNF(C)$
$NNF(\neg\forall R.C) = \exists R.NNF(\neg C)$
$NNF(\neg\exists R.C) = \forall R.NNF(\neg C)$

Algorithme [11]

Soit $K = (T, A)$ et A est consistante.
1 S ← {A}.
2 S'il y a une règle à appliquer Aller vers l'étape 3.
  Sinon K est consistante ; Exit.
3 Appliquer la règle sur S.
4 Supprimer tous les **A'** contenant C(a) et $\neg C(a)$.
5 Si $S = \emptyset$ alors K est inconsistante ; Exit.
  Sinon aller à l'étape 2.

Règles d'extension [11]
Soit A'∈ S :

Règle d'instanciation
Condition :
$C \in T$
Action :
Ajouter C(a) à A' tel que : a est un individu connu (*mentionné dans A'*)
$\cap -$***Règle***
Condition :

$(C \cap D)$ (a) $\in \mathbf{A'}$ et que $C(a)$ et $D(a)$ n'appartiennent pas ***à la fois*** à $\mathbf{A'}$

Action :

Supprimer A' et ajouter A'∪{C (a), D(a)} à S

$\cup -$ **Règle**

Condition :

(C∪ D) (a) $\in \mathbf{A'}$ et que **ni** $C(a) \in \mathbf{A'}$ **ni** $D(a) \in \mathbf{A'}$

Action :

Supprimer A' de S ensuite ajouter A'∪{C(a)} et A' ∪{D(a)} à S.

$\exists -$***Règle***

Condition :

$\exists\, R.C(a) \in A'$ mais qu'il n'existe pas un individu z tel que: $C(z) \in A'$ et $R(a,z) \in A'$

Action :

Supprimer A' est ajouter A'∪{C(z), $R(a,z)$}

$\forall -$***Règle***

Condition :

$\forall\, R.C(a) \in A'$ et $R(a,y) \in A'$ mais que $C(y) \notin A'$

Action :

Supprimer A' est ajouter A'∪{C(y)}

Exemple

Considérons :

- Les concepts atomiques : P, E, M, R
- $K = (T, A)$ tel que $\boldsymbol{T} = \{\boldsymbol{P} \sqsubseteq (\boldsymbol{E} \cap \boldsymbol{M}) \cup (\boldsymbol{E} \sqcap \neg \boldsymbol{R})\}$
- **A=** $\emptyset$

Question : Peut-on inférer à partir de K que $P \sqsubseteq E$?

Pour répondre à cette question on doit démontrer que : $\boldsymbol{K} \cup \{(\boldsymbol{P} \cap \neg \boldsymbol{E})(\boldsymbol{a})\} \models \bot$

Initialisation

S ←{{$(P \cap \neg E)(a)$}}

(∩ −Règle)

S ←{{$(P \cap \neg E)(a), P(a), \neg E(a)$}}

(Règle d'instanciation)

S ←{{$(P \cap \neg E)(a), P(a), \neg E(a), (\neg P \cup (E \cap M) \cup (E \sqcap \neg R))$(a)}}

(∪ − Règle)

S ←{ {$(P \cap \neg E)(a)$, ***P(a)***, $\neg E(a), \big(\neg P \cup (E \cap M) \cup (E \sqcap \neg R)\big)$(a), **¬P(a)**} ,

{$(P \cap \neg E)(a), P(a), \neg E(a), \big(\neg P \cup (E \cap M) \cup (E \sqcap \neg R)\big)$(a), $(E \cap M)$(a)} ,

{$(P \cap \neg E)(a), P(a), \neg E(a), \big(\neg P \cup (E \cap M) \cup (E \sqcap \neg R)\big)$(a), $(E \sqcap \neg R)$(a)}}

(∩ −Règle)

S ←{{$(P \cap \neg E)(a), P(a)$, ***¬E(a)***, $\big(\neg P \cup (E \cap M) \cup (E \sqcap \neg R)\big)$(a), $(E \cap M)$(a), **E(a)**,M(a)} ,{$(P \cap \neg E)(a), P(a)$, ***¬E(a)***, $\big(\neg P \cup (E \cap M) \cup (E \sqcap \neg R)\big)$(a), $(E \sqcap \neg R)$(a),**E(a)**, $\neg R$ (a)} }

S←∅

Conclusion

S=∅ Alors K est Inconsistante et donc **K ⊨ *P* ⊑ *E* (par réfutation)**

2.7 Phase d'opérationnalisation – Langage OWL

Selon la consistance de l'ontologie, on doit choisir le dialecte OWL adéquat en termes d'expressivité. En effet, OWL contient trois sous-langages: OWL Lite, OWL – DL et OWL Full.

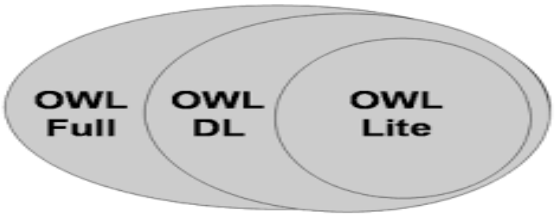


Figure 2.4 Les langages OWL.

Cependant, il y a un compromis entre décidabilité (calculabilité des inférences en un temps fini) et expressivité [10]:

Langage	Expressivité	Décidable
OWL-Lite	Limitée	Oui
OWL-DL	Très bonne	Oui
OWL-Full	Elevée	Non

Citons quelques particularités/limitations de ces langages [10] :

- OWL-Lite est basé sur la logique *SHIF(D)* (*D vient de Datatype*) : il ne permet alors que la définition des rôles fonctionnelles (seulement les cardinalité 0 et 1 sont permises).
- OWL-Lite ne permet pas l'énumération : il interdit alors l'usage du constructeur owl :oneOf.
- OWL-DL est basé sur la logique SHOIN (D). Dans cette logique, les attributs sont considérés comme des rôles (l'ensemble d'arrivée n'est rien autre que le type de l'attribut).
- OWL-Full n'est pas basé sur la logique de description.
- OWL-DL et OWL-Full utilisent les mêmes constructeurs mais OWL-DL impose certaines contraintes sur leurs utilisations. Par exemple, dans OWL-DL :
  - Les classes, les rôles et les individus doivent être disjoints (mais dans OWL-Full par exemple une classe peut être en même temps un individu.)
  - Les constructeurs : *inverseOf, SymmetricProperty et TransitiveProperty* ne peuvent être appliqués sur les *DataTypeProperty* puisque DataTypeProperty est disjoint d'ObjectProperty. Cela reste faisable en langage OWL-Full.

Les tableaux suivants introduisent la syntaxe OWL avec son équivalent en Logique de Description :

Abstract Syntax	DL Syntax
Descriptions (C)	
A	A
`owl:Thing`	$\top$
`owl:Nothing`	$\bot$
`intersectionOf`$(C_1 \dots C_n)$	$C_1 \sqcap \dots \sqcap C_n$
`unionOf`$(C_1 \dots C_n)$	$C_1 \sqcup \dots \sqcup C_n$
`complementOf`(C)	$\neg C$
`oneOf`$(o_1 \dots o_n)$	$\{o_1\} \sqcup \dots \sqcup \{o_n\}$
`restriction`$(R$ `someValuesFrom`$(C))$	$\exists R.C$
`restriction`$(R$ `allValuesFrom`$(C))$	$\forall R.C$
`restriction`$(R$ `hasValue`$(o))$	$R : o$
`restriction`$(R$ `minCardinality`$(n))$	$\geqslant n\, R$
`restriction`$(R$ `minCardinality`$(n))$	$\leqslant n\, R$
`restriction`$(U$ `someValuesFrom`$(D))$	$\exists U.D$
`restriction`$(U$ `allValuesFrom`$(D))$	$\forall U.D$
`restriction`$(U$ `hasValue`$(v))$	$U : v$
`restriction`$(U$ `minCardinality`$(n))$	$\geqslant n\, U$
`restriction`$(U$ `maxCardinality`$(n))$	$\leqslant n\, U$
Data Ranges (D)	
D	D
`oneOf`$(v_1 \dots v_n)$	$\{v_1\} \sqcup \dots \sqcup \{v_n\}$
Object Properties (R)	
R	R
`inv`(R)	R^-
Datatype Properties (U)	
U	U
Individuals (o)	
o	o
Data Values (v)	
v	v

Tableau 2.2 Correspondance entre la syntaxe OWL et la syntaxe LD (partie 1) [12].

Abstract Syntax	DL Syntax
Class(A partial $C_1 \dots C_n$)	$A \sqsubseteq C_1 \sqcap \dots \sqcap C_n$
Class(A complete $C_1 \dots C_n$)	$A \equiv C_1 \sqcap \dots \sqcap C_n$
EnumeratedClass(A $o_1 \dots o_n$)	$A \equiv \{o_1\} \sqcup \dots \sqcup \{o_n\}$
SubClassOf(C_1 C_2)	$C_1 \sqsubseteq C_2$
EquivalentClasses($C_1 \dots C_n$)	$C_1 \equiv \dots \equiv C_n$
DisjointClasses($C_1 \dots C_n$)	$C_i \sqcap C_j \sqsubseteq \bot, i \not\equiv j$
Datatype(D)	
ObjectProperty(R super(R_1)...super(R_n)	$R \sqsubseteq R_i$
domain(C_1)...domain(C_m)	$\geqslant 1\, R \sqsubseteq C_i$
range(C_1)...range(C_ℓ)	$\top \sqsubseteq \forall R.C_i$
[inverseOf(R_0)]	$R \equiv R_0^-$
[Symmetric]	$R \equiv R^-$
[Functional]	$\top \sqsubseteq \leqslant 1\, R$
[InverseFunctional]	$\top \sqsubseteq \leqslant 1\, R^-$
[Transitive])	$Tr(R)$
SubPropertyOf(R_1 R_2)	$R_1 \sqsubseteq R_2$
EquivalentProperties($R_1 \dots R_n$)	$R_1 \equiv \dots \equiv R_n$
DatatypeProperty(U super(U_1)...super(U_n)	$U \sqsubseteq U_i$
domain(C_1)...domain(C_m)	$\geqslant 1\, U \sqsubseteq C_i$
range(D_1)...range(D_ℓ)	$\top \sqsubseteq \forall U.D_i$
[Functional])	$\top \sqsubseteq \leqslant 1\, U$
SubPropertyOf(U_1 U_2)	$U_1 \sqsubseteq U_2$
EquivalentProperties($U_1 \dots U_n$)	$U_1 \equiv \dots \equiv U_n$
AnnotationProperty(S)	
OntologyProperty(S)	
Individual(o type(C_1)...type(C_n)	$o \in C_i$
value(R_1 o_1)...value(R_n o_n)	$\langle o, o_i \rangle \in R_i$
value(U_1 v_1)...value(U_n v_n))	$\langle o, v_i \rangle \in U_i$
SameIndividual($o_1 \dots o_n$)	$\{o_1\} \equiv \dots \equiv \{o_n\}$
DifferentIndividuals($o_1 \dots o_n$)	$\{o_i\} \sqsubseteq \neg\{o_j\}, i \not\equiv j$

Tableau 2.3 Correspondance entre la syntaxe OWL et la syntaxe LD (partie 2) [12].

2.8 Conclusion

A l'issue de ce chapitre, l'étudiant est initié à l'ingénierie ontologique qui est une discipline à part entière. L'intérêt d'une ontologie réside dans le fait qu'elle est partagée (*produit d'un consensus*) et formelle (*compréhensible par la machine*). Néanmoins, la construction d'une ontologie reste une tâche longue et fastidieuse. Actuellement, l'ingénierie ontologique bénéficie des avancements en matière de traitement automatique de langage naturel (*voir le chapitre 5 pour une introduction*). Des outils qui aident à la construction automatique d'ontologies à partir de ressources textuelles existent déjà : *Text2Onto*[3] *et SPRAT*[4] sont des exemples.

La connaissance dont on dispose sur le monde est généralement incertaine. Il est important de pouvoir représenter/raisonner sur des connaissances incertaines. Le chapitre suivant porte sur le raisonnement approximatif via des systèmes d'inférence floue. Pour celui qui a pensé intuitivement à une ontologie-floue: oui cela existe. Pour plus d'approfondissement, la référence [13] constitue un bon point de départ.

[3] http://neon-toolkit.org/wiki/1.x/Text2Onto.html
[4] https://gate.ac.uk/projects/neon/sprat.html

Volet des Travaux Dirigés

TD 1. Conceptualisation d'une première Ontologie

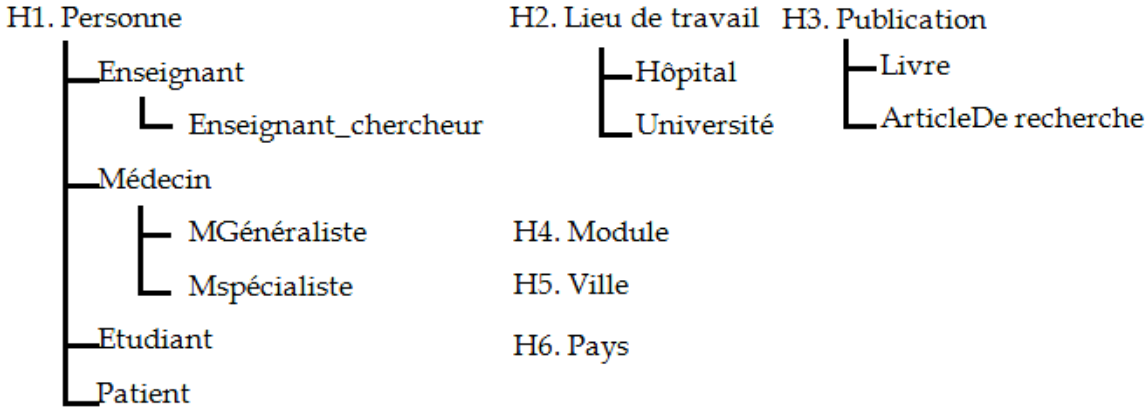
La première phase dans le cycle de vie d'une ontologie est la **conceptualisation**. Dans ce TD, nous construisons le modèle conceptuel d'une ontologie très simple[5] en partant du glossaire des termes candidats suivant:

Terme	Description (informelle)
Personne	Un être humain.
Habite	Une personne habite une ville
Enseignant	Une personne exerçant le métier d'enseignement à l'université
Est Chargé	Un enseignant est chargé d'un module ou plusieurs
Médecin	Une personne qui est docteur en médecine travaillant à l'hôpital : spécialiste ou généraliste
Soigne	Un médecin soigne un ou plusieurs malades
Etudiant	Personne engagé dans un cursus d'enseignement supérieur.
Patient	Une personne qui a été hospitalisée.
Ville	Un milieu urbain où se concentre une forte population humaine (plus de 20 000 habitants) et localisée dans un Pays.
Module	Matière d'enseignement.
Publication	une publication peut être un livre ou un article de recherche.
Ecrit	Une personne peut écrire un livre et les enseignants-chercheurs peuvent écrire des articles de recherche.
….	…

On peut ajouter : Lieu de travail, Livre, Article de recherche, Hôpital, Université, Pays, travaille à…etc.

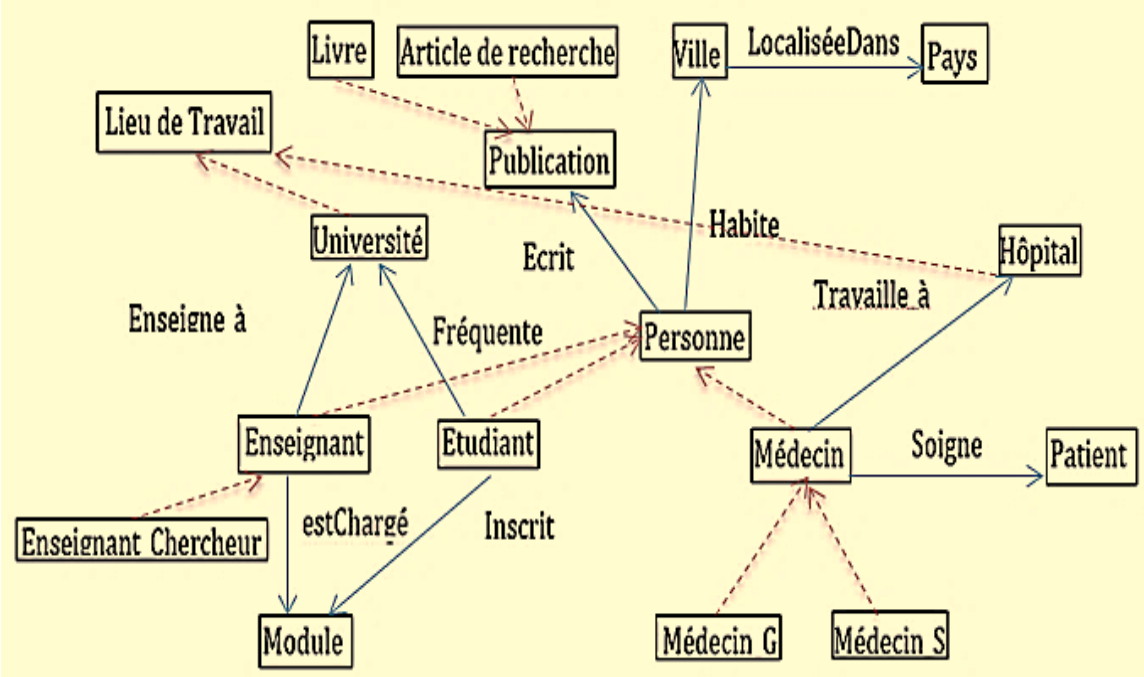
Nous allons suivre les étapes de la méthode METHONTOLOGY déjà vu au cours.

Etape 1 (Hiérarchies de concepts)



[5] Cette ontologie est construite pour des fins pédagogiques, d'où l'absence de la phase d'acquisition des connaissances

Etape 2 (Diagramme de relations binaires)



Etape 3 (Dictionnaire de Concepts)

Concept	*Attribut*	*Synonyme*
Personne	Nom de famille, Prénom Date de naissance	Humain
Module	Intitulé	
Enseignant	Grade	
Enseignant_Chercheur		
Médecin		Docteur en Médecine
Médecin_G		
Médecin_S	Spécialité	
Lieu de Travail	Nom, Secteur	
Hôpital		
Université		
Etudiant	Numéro d'inscription	
Patient	Date d'hospitalisation	Malade
Ville	Nom, Surface	
Pays	Nom, Code_ISO	
Publication	Titre Nombre page	
Livre		
Article de recherche		

Etape 4 (Table des relations binaires)

Relation	*Concept source*	*Concept cible*	*Card Cible*	*Card Source*	*Relation inverse*
Habite	Personne	Ville	*1..1*	200000..n	Habité_par
LocaliséeDans	Ville	Pays	*1..1*	1..n	Contient
Ecrit	Personne	Publication	*0..n*	1..n	Ecrit_Par
estChargé	Enseignant	Module	*1..3*	1..1	Enseigné_par
Enseigne_à	Enseignant	Université	*1..1*	1..n	aPourEnseignant
Travaille_à	Médecin	Hôpital	*1..1*	1..n	aPourMédecin
Soigne	Médecin	Patient	*1..n*	1..n	SoignéPar
Fréquente	*Etudiant*	Université	*1..1*	1..n	*Fréquenté_par*
Inscrit	Etudiant	module	*1..n*	1..n	*Etudié_par*
…	…	….	...	…	....

Etape 5 (Table des attributs)

Attribut	*Type*	*Cardinalité*	*Domaine de valeurs*
Nom de famille	*String*	*1..1*	
Prénom	*String*	*1..2*	
date de naissance	*Date*	*1..1*	
Intitulé	*String*	*1..1*	
Grade	*String*	*1..1*	*MAB ,MAA, MCA,MCB,Prof*
Nom	*String*	*1..1*	
Secteur	*String*	*1..1*	*Santé, EnseignementSup*
Numéro d'inscription	*String*	*1..1*	
Date d'hospitalisation	*Date*	*1..n*	
Surface	*Integer*	*1..1*	
Code	*String*	*1..1*	
Titre	*String*	*1..1*	
Nombre page	*Integer*	*1..1*	

Etape 6 (Liste des axiomes)

Axiome en langage naturelle	*Concepts*	*Relations*
Une personne est soit un étudiant, un enseignant un médecin ou un patient	*Personne* *Etudiant* *Medecin* *Enseignant* *Patient*	*Est-un*
Un médecin est soit un médecin spécialiste ou généraliste	*Medecin* *MedecinS* *MedecinG*	*Est-un*
Un lieu de travail est soit un hôpital ou une université	*Lieu de travail* *université*	*Est-un*
Une publication est soit un livre ou un article de recherche	*Publi* *Livre* *Article*	*Est-un*
Un livre est écrit par au moins une personne	*Livre* *Personne*	*écritPar*
Un article de recherche est écrit par au moins un enseignant-chercheur	*Article de recherche* Enseignant-chercheur	*écritPar*
Un module est enseigné par un et un seul enseignant	*Module* *Enseignant*	*enseignéPar*

Un étudiant est inscrit dans au moins un module et fréquente une université	*Etudiant* *Université* *Module*	*Inscrit* *fréquente*
Une personne habite une et une seule ville	*Personne* *Ville*	*Habite*
Une ville est localisée dans un seul pays	*Ville* *Pays*	*localiséDans*
….	…	**….**

Etape 8 (Table des instances)

Nom de concept	Nom instance	Attribut	valeur
Patient	P_1	Nom de famille Prénom date de naissance date hospitalisation	f Lila 11-07-2005 11-10-2015
MedecinG	MG_1	Nom de famille Prénom date de naissance	ff Ali 15-05-1960
Hôpital	H_1	Nom Secteur	*MdSedikBenYahia* Santé
Ville	V_1	Nom Surface	Jijel 2577 Km2
Pays	Pa_1	Nom CodeISO	Algérie DZ

Etape 8 (Table des assertions)

Nom de relation	Instance source	Instance Cible
Soigne	MG_1	P_1
Habite	MG_1	V_1
	P_1	V_1
TravailleA	MG_1	H_1
LocaliséDans	V_1	Pa_1

TD2. Formalisation de notre Ontologie en LD SHOIN(D)

Construction de la TBOX

$Secteur \equiv \{Santé\} \cup \{EnseignementSup\}$

$Grade \equiv \{MAB\} \cup \{MAA\} \cup \{MCB\} \cup \{MCA\} \cup \{Prof\}$

$LieuTravail \equiv (Hôpital \cup Université) \sqcap{=}\ 1$ Nom. String $\sqcap = 1$ Secteur

$Hôpital \sqsubseteq Lieu\_Travail$

$Université \sqsubseteq LieuTravail$

$Hôpital \sqcap$ Université $\sqsubseteq \perp$

$Module \equiv T \sqcap \exists$ EtudiéPar . Etudiant $\sqcap{=}\ 1$ EnseignéPar . $Enseignant$
$\sqcap{=}\ 1$ Intitulé. String

$Pays \sqcap$ Ville $\sqsubseteq \perp$

$Médecin \equiv (MédecinG \cup MédecinS) \sqcap \exists$ Soigne. Patient $\sqcap \exists$ TravailleA. Hopital

$Personne \equiv (Enseignant \cup Etudiant \cup Médecin \cup Patient) \sqcap \geq 0$ Ecrit. Livre $\sqcap$
$= 1$ Habite. Ville $\sqcap{=}\ 1$ Nom. String $\sqcap{=}\ 1$ dateNaissance. Date $\sqcap$
≥ 1 Prénom. String $\sqcap{\leq}\ 2$ Prénom. String

(… On fait de même pour tous les concepts)

$Ecrit \equiv (EcritPar)^{-}$

$Enseigne \equiv (EnseignéPar)^{-}$

(… On fait de même pour tous les rôles /rôles inverses)

Construction de l'ABOX

$Hôpital\ (H\_1)$

$Patient\ (P\_1)$

$MédecinG(MG\_1)$

(… On fait de même pour tous les individus ou instances)

$Soigne(MG\_1, P\_1)$

$SoignéPar(P\_1, MG\_1)$

…

$Nom\ (H\_1, MdSedikBenYahia)$

$Secteur(H\_1, Santé)$

(… On fait de même pour toutes les assertions)

TD 3. Opérationnalisation de l'ontologie avec PROTéGé

Le codage de l'ontologie dans un langage opérationnel (OWL dans notre contexte) se fait pratiquement à l'aide d'un éditeur d'ontologie. En effet, PROTéGé est le logiciel le plus utilisé. Il est téléchargeable à l'adresse ci-dessous :

http://protege.stanford.edu/download/protege/4.3/installanywhere/Web_Installers/

Remarque: Une Séance de TP doit être prévue pour éditer l'ontologie avec PROTEGE.

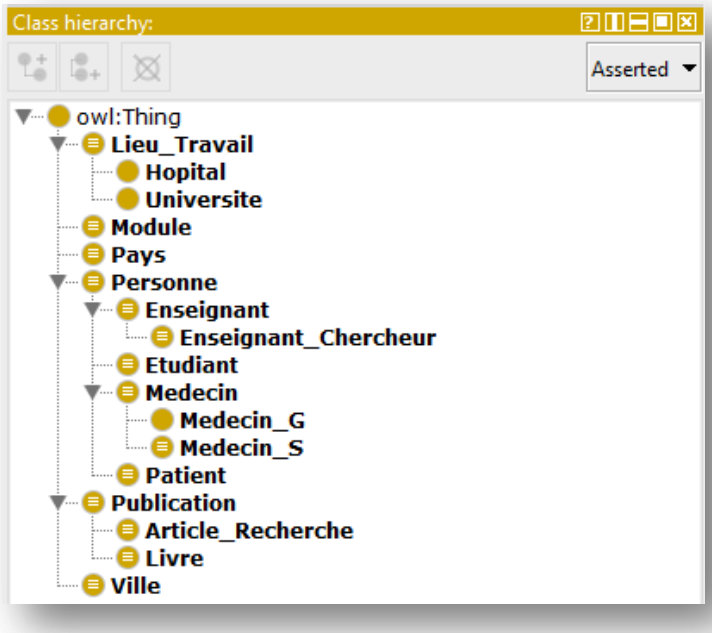


Figure 2.5 Hiérarchie de classe sous Protégé.

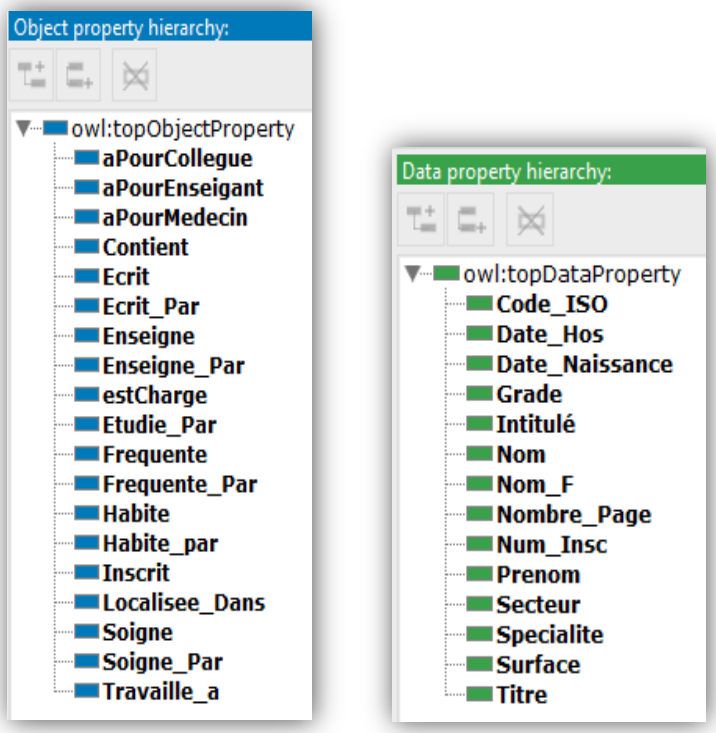


Figure 2.6 Hiérarchie des relations binaires et des attributs sous Protégé.

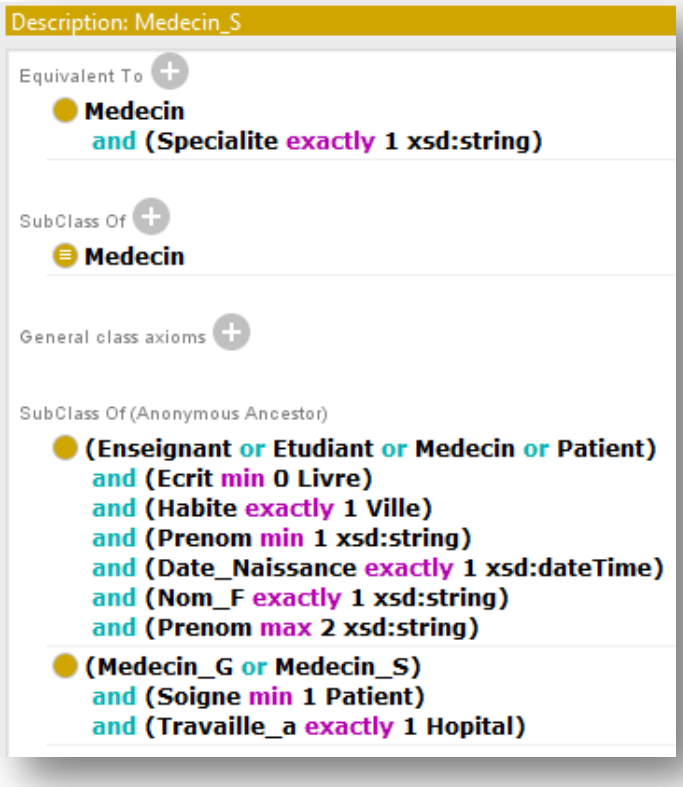


Figure 2.7 Définition du concept Medecin_S dans Protégé

Dans ce TD, le code OWL généré automatiquement par l'outil PROTEGE est décortiqué.
Un document OWL est un document RDF avec l'élément racine rdf:RDF :

<**rdf:RDF**
xmlns ="**http://www.example.org/**"
xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:xsd ="http://www.w3.org/2001/XMLSchema#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl ="http://www.w3.org/2002/07/owl#">
L'élément owl : Ontology donne des informations sur l'ontologie :
<**owl:Ontology**
rdf:about="http://www.semanticweb.org/toshiba/ontologies/2016/3/exemple">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
Ontologie pour TD version : Mai 2016
</rdfs:comment>
</owl:Ontology>

[Les rôles : *Object Properties and Data Properties*]

```
<owl:DatatypeProperty rdf:about="Date_Hos">
<rdfs:domain rdf:resource="Patient"/>
<rdfs:range rdf:resource="&xsd;dateTime"/>
</owl:DatatypeProperty>
```

```
<owl:DatatypeProperty rdf:about="Code_ISO">
<rdfs:domain rdf:resource="Pays"/>
<rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="Secteur">
<rdfs:domain rdf:resource="Lieu_Travail"/>
<rdfs:range>
<rdfs:Datatype>
<owl:oneOf>
<rdf:Description>
<rdf:type rdf:resource="&rdf;List"/>
<rdf:first>Enseignement_Sup</rdf:first>
<rdf:rest>
<rdf:Description>
<rdf:type rdf:resource="&rdf;List"/>
<rdf:first>Sante</rdf:first>
<rdf:rest rdf:resource="&rdf;nil"/>
</rdf:Description></rdf:rest></rdf:Description>
</owl:oneOf>
</rdfs:Datatype>
</rdfs:range></owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="Nom">
<rdfs:range rdf:resource="&xsd;string"/>
<rdfs:domain>
<owl:Class>
<owl:unionOf rdf:parseType="Collection">
<rdf:Description rdf:about="Lieu_Travail"/>
<rdf:Description rdf:about="Pays"/>
<rdf:Description rdf:about="Ville"/>
</owl:unionOf>
</owl:Class>
</rdfs:domain>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:about="estCharge">
<rdfs:domain rdf:resource="Enseignant"/>
<rdfs:range rdf:resource="Module"/>
<inverseOf rdf:resource="Enseigne_Par"/>
 </owl :ObjectProperty>
<owl: ObjectProperty rdf:about="Enseigne_Par">
<rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
</owl: ObjectProperty>
```

Question : Ajouter la relation **a pour collègue** à l'ontologie.

Réponse

```
<owl:ObjectProperty rdf:about="aPourCollegue">
<rdfs:domain rdf:resource="Personne"/>
```

```
<rdfs:range rdf:resource="Personne"/>
<rdf:type rdf:resource="&owl;TransitiveProperty" />
<rdf:type rdf:resource="&owl;SymmetricProperty" />
</owl:ObjectProperty>
```

[Les Concepts : *Classes*]

```
<owl:Class rdf:about="Medecin_G">
<rdfs:subClassOf rdf:resource="Medecin"/>
<owl:disjointWith rdf:resource="Medecin_S"/>
</owl:Class>
<owl:Class rdf:about="Etudiant">
<owl:equivalentClass>
<owl:Class>
<owl:intersectionOf rdf:parseType="Collection">
<rdf:Description rdf:about="Personne"/>
<owl:Restriction>
<owl:onProperty rdf:resource="Inscrit"/>
<owl:onClass rdf:resource="Module"/>
<owl:minQualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger"> 1
</owl:minQualifiedCardinality>
</owl:Restriction>
<owl:Restriction>
<owl:onProperty rdf:resource="Frequente"/>
<owl: onClass rdf:resource="&exemple;Universite"/>
 <qualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger"> 1
</qualifiedCardinality>
</owl:Restriction>
<owl:Restriction>
<owl:onProperty rdf:resource="Num_Insc"/>
<qualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger"> 1
</qualifiedCardinality>
<owl:onDataRange rdf:resource="&xsd;string"/>
</owl:Restriction>
</owl:intersectionOf>
</owl:Class>
</owl:equivalentClass>
</owl:Class>
```

Question. Définir le **concept Ville** en OWL.

Réponse

```
<owl:Class rdf:about=”Ville">
<owl:equivalentClass>
<owl:Class>
<owl:intersectionOf rdf:parseType="Collection">
<rdf:Description rdf:about="&owl;Thing"/>
<owl :Restriction>
```

```
<owl :onProperty rdf:resource="Habite_par"/>
<owl :onClass rdf:resource="Personne"/>
<owl :minQualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">200000 </owl:
minQualifiedCardinality>
</owl : Restriction>
<owl : Restriction>
<owl: onProperty rdf:resource="Localisee_Dans"/>
<owl: onClass rdf:resource="Pays"/>
<owl: qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</qualifiedCardinality>
</owl:Restriction>
<owl: Restriction>
<onProperty rdf:resource="Nom"/>
<qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</qualifiedCardinality>
<owl:onDataRange rdf:resource="&xsd;string"/>
</owl:Restriction>
<owl:Restriction>
<owl:onProperty rdf:resource="&exemple;Surface"/>
<owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</qualifiedCardinality>
<owl:onDataRange rdf:resource="&xsd;positiveInteger"/>
</owl:Restriction>
</owl:intersectionOf>
</owl:Class>
</owl:equivalentClass>
</owl:Class>
```

[Les instances/assertions : *Individuals*]

```
<owl:NamedIndividual rdf:about="Pa_1">
<Nom rdf:datatype="&xsd;string">Algerie</Nom>
<Code_ISO rdf:datatype="&xsd;string">DZ</Code_ISO>
</owl:NamedIndividual>
<owl:NamedIndividual rdf:about="MG1">
<Date_Naissance rdf:datatype="&xsd;dateTime">15-05-1960</Date_Naissance>
<Prenom rdf:datatype="&xsd;string">Ali</Prenom>
<Nom_F rdf:datatype="&xsd;string">ff</Nom_F>
<Travaille_a rdf:resource="H1"/>
<Soigne rdf:resource="P1"/>
<Habite rdf:resource="V1"/>
</owl:NamedIndividual>
```

TD4. Logiques de Description [14]

Exercice 1

1) Pour une interprétation I est un rôle R, combien de paires d'éléments peut-on avoir dans R^I ?
2) Pour un élément $e \in \Delta^I$, peut-on avoir $(e, e) \in R^I$?
3) Pour deux éléments $e, f \in \Delta^I$, peut-on avoir $\{(e, f), (f, e)\} \subseteq R^I$?
4) Soit la TBOX

$\{Enseignant \sqsubseteq Personne,$
$CoursMaster \sqsubseteq \neg Personne,$
$Enseignant \sqsubseteq \exists Enseigne.Cours,$
$\exists Enseigne.Cours \sqsubseteq Personne\}$

Vérifier si l'interprétation suivante est un modèle pour cette TBOX:

$\Delta^I = \{m, c6, c7, et\}$
$Enseignant^I = \{m\}$
$Cours^I = \{c6, c7, et\}$
$Personne^I = \{m, et\}$
$CoursMaster^I = \{c7\}$
$Enseigne^I = \{(m, c6), (m, c7), (et, et)\}$

Refaire la question précédente si on ajoute à la TBOX les axiomes:

$$\{Cours \sqsubseteq \neg Personne, \exists Enseigne.Cours \sqsubseteq Enseignant\}$$

Exercice 2

Utiliser les concepts atomiques : Personne, Heureux, Animal, Chat, Vieux, Poisson et le rôle atomique Possède pour définir en logique ALC :

- Personne heureuse
- Personne heureuse qui possède un animal
- Personne qui possède seulement des chats
- Personne Malheureuse qui possède un vieux chat
- Personne qui possède seulement des chats ou des poissons.

Exercice 3

1) Les Concepts suivants sont-ils satisfiables?

$A \cup \neg A$
$A \sqcap \exists R.B \sqcap \forall S.\neg B$
$A \sqcap \exists R.B \sqcap \forall R.\neg B$
$A \sqcap \exists R.(B \cap C) \sqcap \forall R.\neg B$

2) Lequel des énoncés suivants est vrai ?

B est subsumé par$A \cup \neg A$
$A \sqcap \exists R.B$ est subsumé par $A \sqcap \exists R.T$
$A \sqcap \exists R.(B \cup C)$ est subsumé par $A \sqcap \exists R.B$
$A \sqcap \exists R.B$ est subsumé par $A \sqcap \forall R.B$
$A \sqcap \exists R.A \sqcap \forall R.B$ est subsumé par $A \sqcap \exists R.B$

Exercice 4

Exprimer en langage naturelle les axiomes LD suivants :

$\text{Conducteur} \sqcap \exists \text{Controle}.\text{Voiture} \sqsubseteq \text{Majeur}$

$\text{Vélo} \sqsubseteq (= 2\ \text{Contient}.\text{Roue})$

$\text{Voiture} \sqsubseteq (= 1\ \text{Controle}^{-}.\text{Humain})$

Tr(Contient)

Donner l'équivalent en logique des prédicats des axiomes LD suivants :

$\text{Humain} \sqsubseteq \neg \text{Voiture}$

$\exists \text{Controle}.\text{Voiture}\ (\text{Bob})$

$\text{Conducteur} \equiv \text{Humain} \sqcap \exists \text{Controle}.\text{Véhicule}$

$\text{Vélo} \equiv \text{Véhicule} \cap \exists \text{Contient}.\text{Roue} \cap \exists \text{conduitePar}.\text{Humain}$

Exercice 5

Étant donné le *fragment de code OWL* suivant, donner son équivalent en LD.

```
1.  <owl:Class rdf:about="Livre">
2.  <rdfs:subClassOf rdf:resource="Publication"/>
3.  <owl:disjointWith rdf:resource="Magazine"/>
4.  </owl:Class>
5.  <owl:Class rdf:about="Magazine">
6.  <rdfs:subClassOf rdf:resource="Publication"/>
7.  </owl:Class>
8.  <owl:Class rdf:about="Publication">
9.  <owl:equivalentClass>
10. <owl:Class>
11. <owl:unionOf rdf:parseType="Collection">
12. <rdf:Description rdf:about="Livre"/>
13. <rdf:Description rdf:about="Magazine"/>
14. </owl:unionOf>
15. </owl:Class>
16. </owl:equivalentClass>
17. </owl:Class>
18. <owl:NamedIndividual rdf:about="Per_11">
19. <rdf:type rdf:resource="Personne"/>
20. </owl:NamedIndividual>
```

Références du chapitre

[1] G. Caplat, *Modélisation cognitive et résolution de problèmes* ;PPUR presses polytechniques, 2002.

[2] A. Newell, *The knowledge level*, Artificial Intelligence, vol. 18, no. 1, pp. 87-127, 1982.

[3] M. Madeleine V. Pietri, *L'ingénierie de la connaissance: la nouvelle épistémologie appliquée*, Volume 696, Presses Univ. Franche-Comté, 2000.

[4] P. Serrafero, *Cycle de vie, maturité et dynamique de la connaissance : des informations aux cognitions de l'entreprise Apprenante* , Revue annuelle U.E. des Arts et Métiers sur le Knowledge Mangement, Edition Dunod, 2000.

[5] T.R. Gruber, *A Translation Approach to Portable Ontology Specifications.* Knowledge Acquisition: Vol.5, n.2, pp. 199–220, 1993.

[6] M. Hadzic, *Ontology-based multi-agent systems*, Springer, 2014.

[7] M. Fernandez-Lopez, A. Gomez-Pérez, N. Juristo, *METHONTOLOGY: From Ontological Art Towards Ontological Engineering*. Spring Symposium on Ontological Engineering of AAAI. Stanford University, California, pp 33–40, 1997.

[8] A. Gomez-Perez, M. Fernandez-Lopez and O. Corcho, *Ontological engineering*. London: Springer-Verlag, 2010.

[9] F. Baader, D. Calvanese, D. McGuinness, D. Nardi and P. Patel-Schneider, *The description logic handbook*. Cambridge: Cambridge University Press, 2010.

[10] P. Hitzler, M. Krötzsch and S. Rudolph, *Foundations of semantic web technologies*. Boca Raton, Fla: CRC, 2010.

[11] G. Groner and M. Thimm, *Semantic Web : Description logics*, notes de cours, Institute for Web Science and Technologies, University of Koblenz-Landau, 2013.

[12] I. Horrocks, *OWL: A Description Logic Based Ontology Language,* In: van Beek P. (eds) Principles and Practice of Constraint Programming. Lecture Notes in Computer Science, vol. 3709. Springer, Berlin, Heidelberg, 2005.

[13] F. Bobillo and U. Straccia, *Fuzzy ontology representation using OWL 2*, International Journal of Approximate Reasoning, vol. 52, no. 7, pp. 1073-1094, 2011.

[14] F. Baader, I. Horrocks, C. Lutz and U. Sattler : *A Basic Description Logic. In An Introduction to Description Logic*, pp. 10-49. Cambridge: Cambridge University Press, 2017.

CHAPITRE 3

Systèmes d'Inférence Floue

3.1 Introduction

L'être humain est capable d'utiliser des connaissances incertaines ou imprécises afin de produire de nouvelles connaissances. La logique floue est un outil de l'intelligence artificielle qui permet à la machine d'imiter cette faculté de raisonnement approximatif. Cependant, il ne faut pas comprendre que « la logique floue est floue ». Bien au contraire : « la logique floue est une logique précise de l'imprécision et du raisonnement approximatif» [1].

La logique floue est basée sur les sous-ensemble-flous : un concept introduit la première fois en 1965 par Lotfi Zadeh [2]. Dans ce chapitre, nous expliquons succinctement les éléments de la logique floue permettant la construction d'un système d'inférence floue (SIF). Particulièrement, nous abordons les SIFs de type MAMDANI et TSK qui sont les plus cités dans la littérature mais aussi qui ont prouvé leur performance dans l'industrie.

Enfin, des exemples portant sur l'utilisation des SIFs dans l'implémentation de différents systèmes intelligents sont présentés. L'objectif étant de faire le lien avec les connaissances déjà acquises, par les étudiants, sur la thématique de *conception des agents intelligents*.

3.2 Logique Floue

Les connaissances issues du monde réel sont typiquement imparfaites. Par imperfection, nous entendons l'incertitude (*par rapport à la validité*) et l'imprécision (*difficulté dans l'expression claire*) des connaissances. L'être humain reste néanmoins capable d'exploiter des perceptions vagues et de prendre par la suite une décision adéquate à la situation rencontrée.

La logique floue permet une modélisation mathématique rigoureuse de l'imprécision/incertitude comme l'illustre les sous-sections suivantes.

3.2.1 Concept d'ensemble flou[1]

Etant donné un univers de discours U (*domaine de variation des variables*), les frontières d'un ensemble classique sont strictement définies. Un objet est soit membre d'un ensemble ou non. Formellement, un ensemble classique peut être noté par $A = \{x \in U | P(x)\}$ où les éléments x de A vérifient la propriété P. La fonction d'appartenance à A est définie comme suit : $\mu_A(x): U \longrightarrow \{0,1\}$.

Prenons l'exemple [3] de l'ensemble des personnes adultes A classifiées selon la propriété Age: $A = \{x \in U | Age(x) \geq 18\}$. Clairement, les personnes âgées de 5 ans ne sont pas des éléments de A contrairement à celles âgées de 45 ans. Un problème se pose quant à la classification des personnes âgées par exemple de « 17 ans et 9 mois » qui, selon la définition de l'ensemble A, ne sont pas des adultes. Il sera cependant plus juste de dire que «les personnes de 17 ans et 9 mois sont presque des adultes». Il est possible d'établir une telle description grâce au concept d'ensemble flou qui permet d'indiquer le degré d'appartenance d'un élément à un ensemble.

On dit que le passage de l'appartenance à la non-appartenance dans un ensemble flou se fait progressivement.

Définition de la fonction d'appartenance d'un ensemble flou

Soit U l'univers de discours. Un ensemble flou A de U est caractérisé par une fonction d'appartenance $\mu_A(x)$ qui associe à chaque point dans U un nombre réel dans l'intervalle [0,1] où la valeur de $\mu_A(x)$ représente le degré d'appartenance de x à A: $\mu_A(x): U \longrightarrow [0,1]$.

Remarque

Un ensemble flou A est ***vide*** SSI sa fonction d'appartenance est identiquement nulle sur U. Plus formellement : $\forall x \in U: \mu_A(x) = 0$.

3.2.2 Degré d'appartenance Vs Probabilité

Afin d'illustrer la différence entre degré d'appartenance et probabilité, nous utilisons l'exemple suivant pris de la référence [4] :

Supposons que l'univers de discours soit l'ensemble de tous les liquides. Soit L l'ensemble flou des liquides potables. Imaginons que vous êtes dans un désert sans rien à boire...vous tombez sur deux bouteilles:

- Bouteille A avec l'indication : degré d'appartenance de A à L = 0.9.
- Bouteille B avec l'indication : probabilité que B appartienne à L=0.9.

[1] Ou sous-ensemble floue (les deux termes sont permis)

Quelle bouteille allez-vous choisir ? La réponse doit être A puisque :

- Pour B : il y a une chance de 10% que le liquide soit dangereux.
- Pour A : le liquide est très proche d'un liquide pur (mais pas à 100% évidement).

3.2.3 Opérations sur les ensembles flous

Les définitions suivantes sont compilées à partir des références [3,4] :

Deux ensembles flous A et B sont ***égaux***, et on écrit $A = B$, SSI :

$$\forall x \in U: \mu_A(x) = \mu_B(x)$$

Un ensemble flou A est ***inclus*** dans B, et on écrit $A \subseteq B$ SSI :

$$\forall x \in U \ \ \mu_A(x) \leq \mu_B(x)$$

Informellement : si pour n'importe quel élément x , x appartient moins à A qu'à B.

L'**union** de deux ensembles flous A et B est caractérisée par la fonction d'appartenance suivante :

$$\mu_{A \cup B}(x) = \text{Max}[\mu_A(x), \mu_B(x)]$$

Intuitivement, un élément ne peut appartenir à l'union de deux ensembles flous plus fortement qu'il n'appartient à chacun d'eux.

L'**intersection** de deux ensembles flous A et B est caractérisée par la fonction d'appartenance suivante :

$$\mu_{A \cap B}(x) = \text{Min}[\mu_A(x), \mu_B(x)]$$

Au fait, un élément ne peut appartenir à l'intersection de deux ensembles flous moins fortement qu'il n'appartient à chacun d'eux.

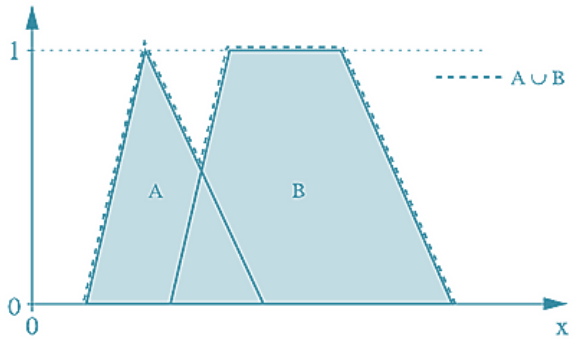


Figure 3.1 Union de deux ensembles flous avec l'opérateur Max [4].

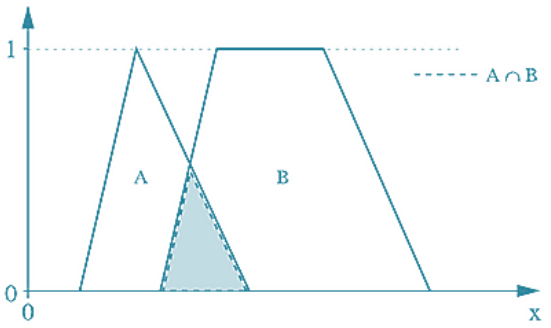


Figure 3.2 Intersection de deux ensembles flous avec l'opérateur Min [4].

Le **complément** d'un ensemble flou A est noté $\bar{A}$ et sa fonction d'appartenance est définie comme suit : $\forall x \in U \ : \ \mu_{\bar{A}}(x) = \ 1 - \mu_A(x)$.

Remarque

La propriété de la non contradiction n'est pas satisfaite pour les ensembles flous ($\bar{A} \cap A \ \neq \emptyset$) et non plus la propriété du tiers exclu ($\bar{A} \cup A \ \neq U$) .

3.2.4 Types des fonctions d'appartenance

Une fonction d'appartenance peut prendre différentes formes comme le montre le tableau suivant :

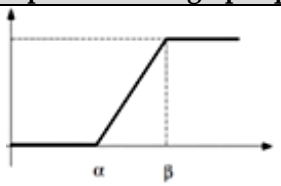
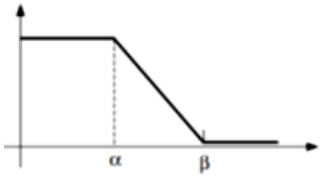
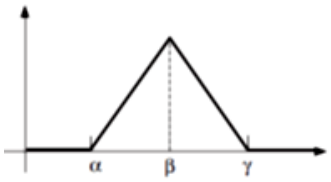
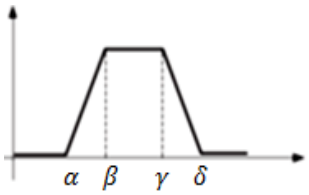
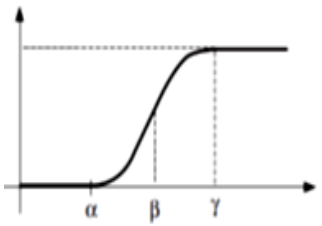
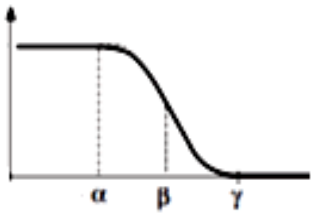
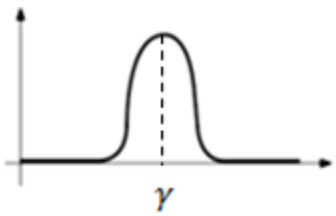
Fonction d'appartenance	Représentation graphique	Définition
Γ –fonction		$\Gamma: U \longrightarrow [0,1]$ $\Gamma(x;\alpha,\beta) = \begin{cases} 0 & x < \alpha \\ \frac{x-\alpha}{\beta-\alpha} & \alpha \le x \le \beta \\ 1 & x > \beta \end{cases}$
L –fonction		$L: U \longrightarrow [0,1]$ $L(x;\alpha,\beta) = \begin{cases} 1 & x < \alpha \\ \frac{x-\beta}{\alpha-\beta} & \alpha \le x \le \beta \\ 0 & x > \beta \end{cases}$
Λ –fonction ou fonction Triangulaire		$\Lambda: U \longrightarrow [0,1]$ $\Lambda(x;\alpha,\beta,\gamma) = \begin{cases} 0 & x < \alpha \\ \frac{x-\alpha}{\beta-\alpha} & \alpha \le x \le \beta \\ \frac{x-\gamma}{\beta-\gamma} & \beta \le x < \gamma \\ 0 & x \ge \gamma \end{cases}$
Π –fonction ou fonction Trapézoïdal		$\Pi: U \longrightarrow [0,1]$ $\Pi(x;\alpha,\beta,\gamma,\delta) = \begin{cases} 0 & x < \alpha \\ \frac{x-\alpha}{\beta-\alpha} & \alpha \le x < \beta \\ 1 & \beta \le x < \gamma \\ \frac{x-\delta}{\gamma-\delta} & \gamma \le x < \delta \\ 0 & x \ge \delta \end{cases}$
S –fonction croissante ou fonction *sigmoïdal croissante*		$S: U \longrightarrow [0,1]$ $S(x;\alpha,\beta,\gamma) = \begin{cases} 0 & x < \alpha \\ 2\left(\frac{x-\alpha}{\gamma-\alpha}\right)^2 & \alpha \le x \le \beta \\ 1-2\left(\frac{x-\alpha}{\gamma-\alpha}\right)^2 & \beta \le x \le \gamma \\ 1 & x > \gamma \end{cases}$
S –fonction décroissante ou fonction *sigmoïdal décroissante*		$S: U \longrightarrow [0,1]$ $S(x;\alpha,\beta,\gamma) = \begin{cases} 1 & x < \alpha \\ 1-2\left(\frac{x-\alpha}{\gamma-\alpha}\right)^2 & \alpha \le x \le \beta \\ 2\left(\frac{x-\alpha}{\gamma-\alpha}\right)^2 & \beta \le x \le \gamma \\ 0 & x > \gamma \end{cases}$
Fonction gaussienne		$G: U \longrightarrow [0,1]$ $G(x;\gamma,\sigma) = \exp(-\frac{(\gamma-x)^2}{2\sigma^2})$ Où σ est l'écart type

Tableau 3.1 Fonctions d'appartenance usuelles [3].

Remarque : Vu leur simplicité, Γ −fonction, L −fonction et Λ − fonction restent les formes les plus utilisées dans la pratique.

3.3 Raisonnement approximatif

Dans cette partie, nous introduisons les concepts de la logique floue nécessaires pour la mise en œuvre d'un raisonnement approximatif par le biais d'un Système d'Inférence Floue (SIF).

3.3.1 Variable linguistique

Une ***variable linguistique*** u est généralement identifiée par un ensemble de termes $T(u)$ couvrant son univers de discours U.

Exemple

Considérons la variable linguistique $Température$ et l'univers de discours $U = [0,40]$.

$T(Température) = \{Faible, Moyenne, Elevée\}$.

A Chaque terme est associé un ensemble flou :

- ✓ F : température inférieure à 15°C
- ✓ M : température proche de 20°C
- ✓ E : température au-delà de 25°C

Les fonctions d'appartenance associées aux ensembles flous F, M, E sont graphiquement présentées dans la figure ci-dessous.

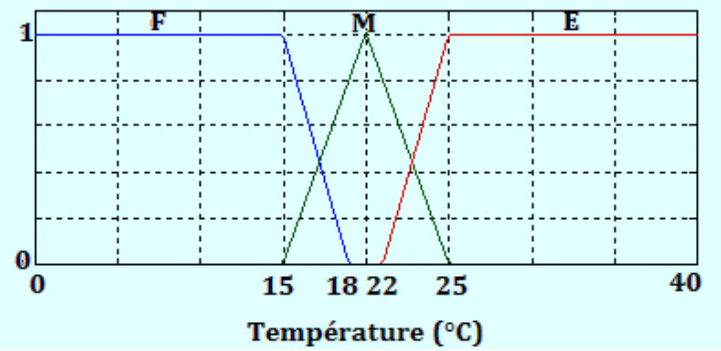


Figure 3.3 Ensembles flous de la variable linguistique Température [3].

Question 1: auxquels ensembles flous appartiennent 17°C et 23°C et avec quel degré d'appartenance ?

- Clairement 17°C est membre des ensembles flous F et M et 23°C est membre des ensembles flous M et E.
- Pour calculer les degrés d'appartenance, nous utilisons les définitions qui conviennent *(selon la forme : triangulaire, trapèze, etc.)* du tableau 3.1.

$$\mu_F(x) = \begin{cases} 1 & x < 15 \\ \dfrac{x-18}{15-18} & 15 \le x \le 18 \\ 0 & x > 18 \end{cases} \qquad \mu_M(x) = \begin{cases} 0 & x < 15 \\ \dfrac{x-15}{20-15} & 15 \le x \le 20 \\ \dfrac{x-25}{20-25} & 20 \le x < 25 \\ 0 & x \ge 25 \end{cases}$$

$$\mu_E(x) = \begin{cases} 0 & x < 22 \\ \dfrac{x - 22}{25 - 22} & 22 \leq x \leq 25 \\ 1 & x > 25 \end{cases}$$

Il ne reste qu'à faire l'application numérique selon la valeur de x.
Ce type de calcul correspond à ***une Fuzzification*** *(dans le sens : passage d'une valeur numérique vers un degré d'appartenance).*

Question 2 : Trouver la température x pour la quelle $\mu_F(x) = 0.25$.

- Il suffit de mettre : $\frac{x-18}{-3} = 0.25$ Ce qui donne x=17.25 °C

Ce type de calcul correspond à ***une Defuzzification*** (*dans le sens : passage d'un degré d'appartenance vers une valeur numérique).*

3.3.2 Proposition floue

Une ***proposition floue*** élémentaire prend la forme « *Terme linguistique* ***est*** *ensemble flou* » [4]. Notons qu'une proposition floue comme : « Température est Moyenne » a pour valeur de vérité le degré d'appartenance de Température à l'ensemble floue M.

3.3.3 Opérations de la logique floue

En appliquant les opérations de la logique floue sur des propositions floues élémentaires, on peut construire des propositions floues générales. Notons que les opérations de la logique floue sont fortement liées aux définitions des opérations sur les ensembles flous.

Opération	Relation	Fonction d'appartenance
Négation	*R : Non A*	$\mu_R(x) = 1 - \mu_A(x)$
Disjonction	*R : A ou B*	$\mu_R(x) = \max\{\mu_A(x), \mu_B(x)\}$
Conjonction	*R : A et B*	$\mu_R(x) = \min\{\mu_A(x), \mu_B(x)\}$
Implication	$R : (x = A) \longrightarrow (y = B)$	*Implication de Zadeh* : $\mu_R(x) = \max\{\min\{\mu_A(x), \mu_B(y)\}, 1 - \mu_A(x)\}$ *Implication de Mamdani :* $\mu_R(x) = \min\{\mu_A(x), \mu_B(y)\}$ *Implication de Larsen :* $\mu_R(x) = \mu_A(x) \times \mu_B(y)$ *Implication de Godel:* $\mu_R(x) = \begin{cases} 1 & si\ \mu_A(x) \leq \mu_B(y) \\ \mu_B(y) & sinon \end{cases}$ *Implication de Lukasiewicz* : $\mu_R(x) = \min\{1, [1 - \mu_A(x) + \mu_B(y)]\}$

Tableau 3.2 Opérations de la logique floue [3].

Remarque
Les définitions de l'implication les plus utilisées dans la pratique sont celles de Mamdani et de Larsen.

3.4 Système d'inférence floue de type MAMDANI

Un *SIF* effectue un passage *non linéaire* d'un *vecteur* de variables numériques, en entrée, à une sortie *scalaire* également numérique. Un SIF de type Mamdani peut-être schématisé comme dans la figure 3.4. A savoir, ce type de SIF a été proposé en 1975 par Ebrahim Mamdani [5] pour le contrôle d'une machine à vapeur (*steam engine*).

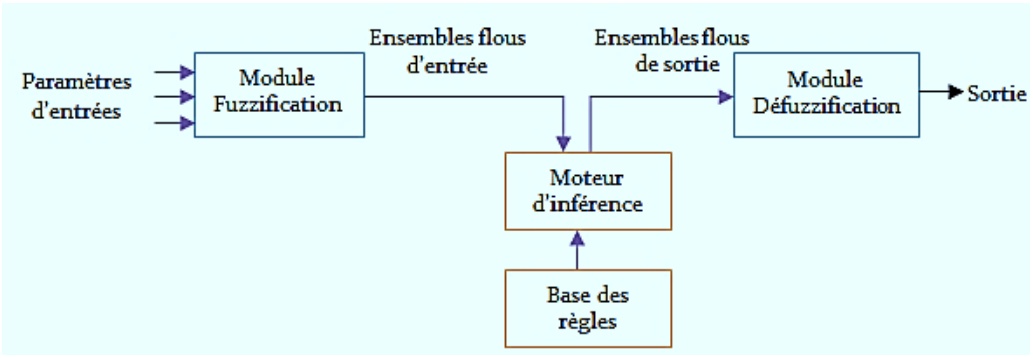


Figure 3.4 Les différents modules d'un SIF de type MAMDANI [3].

Dans ce qui suit, nous expliquons le fonctionnement des différents modules [3] :

- **Module de Fuzzification :** Ce module associe aux paramètres d'entrée numériques des *ensembles flous d'entrée* tout en déterminant le degré d'appartenance à chacun de ces ensembles.
- **Base des règles :** La base des règles représente la stratégie de décision ou de contrôle utilisée par le système. Les règles sont généralement établies par les experts du domaine de l'application envisagée. Une règle possède la forme d'une implication «SI - ALORS». La partie SI de l'implication est appelé *antécédent, condition* ou *prémisse* tandis que la partie ALORS est nommée *conséquent, conclusion ou action*.

  La partie antécédent contient une ou plusieurs propositions floues combinées par des opérateurs flous de conjonction, de disjonction ou de négation. Dans le cas des SIFs structurés selon le *modèle de Mamdani* conventionnelle dit *modèle linguistique*, la forme la plus commune est celle conjonctive. Alors, dans une base composée de M règles, la l^{eme} règle est exprimée comme suit :

$$R^{(l)}: \quad \textbf{SI } u_1 est\ F_1^l\ ET\ u_2 est\ F_2^l\ ET \dots u_p est\ F_p^l\ \textbf{ALORS}\ v\ est\ G^l$$

  Où $u = (u_1, u_2, \dots, u_p) \in U_1 \times \dots \times U_p$ et $v \in V$ sont des variables linguistiques. F_i^l pour tout $i \in \{1..p\}$ et G^l sont des ensembles flous des univers de discours $U_i \subset \mathbb{R}$ et $V \subset \mathbb{R}$, respectivement.

  Notons que les règles d'inférence sont reliées par « *ELSE* » d'où l'application de l'opérateur MAX lors de l'agrégation des sorties floues obtenues [5].
- **Moteur d'inférence:** Etant donnée une collection de règles, le moteur d'inférence a pour rôle d'associer aux entrées numériques du système un seul ensemble flou de sortie. Durant le processus d'inférence, les opérations suivantes sont invoquées :
  a) Calcul du degré d'accomplissement ou d'activation de chaque règle à partir des degrés d'appartenance des variables floues d'entrées. Une règle est activée dès qu'elle a une prémisse ayant une valeur de vérité non nulle.

b) Pour chaque règle activée, trouver la fonction d'appartenance de la conclusion.

c) Agrégation des conclusions inférées.

Plusieurs méthodes d'inférence ont été proposées. Les plus utilisées sont : MAX-MIN (*inférence de Mamdani*), Max-Produit (*inférence de Larsen*) et Somme-Produit. De par sa simplicité, l'inférence de Mamdani est de loin la méthode la plus utilisée. Elle consiste à : utiliser le MIN pour la conjonction et l'implication et le MAX pour la disjonction.

- **Defuzzification:** La sortie d'un SIF est souvent orientée pour être utilisée par une machine. Il convient donc de convertir la sortie floue obtenue à une valeur numérique exploitable : c'est exactement le rôle du module de défuzzification. Plusieurs méthodes de défuzzification ont été proposées, les plus cités dans la littérature sont:

  a) *Le centre de gravité (Centroid en anglais):* cette méthode consiste à prendre comme sortie, l'abscisse du centre de gravité de la fonction d'appartenance de la sortie issue de l'inférence floue.

  b) *Méthode du maximum:* La sortie correspond à l'abscisse du maximum de la fonction d'appartenance résultante. Lorsqu'il existe plusieurs valeurs pour lesquelles la fonction d'appartenance résultante est maximale, on peut opter pour [6]: SOM (*Smallest Of Maximum*), MOM (*Middle Of Maximum*) ou LOM (*Largest Of Maximum*).

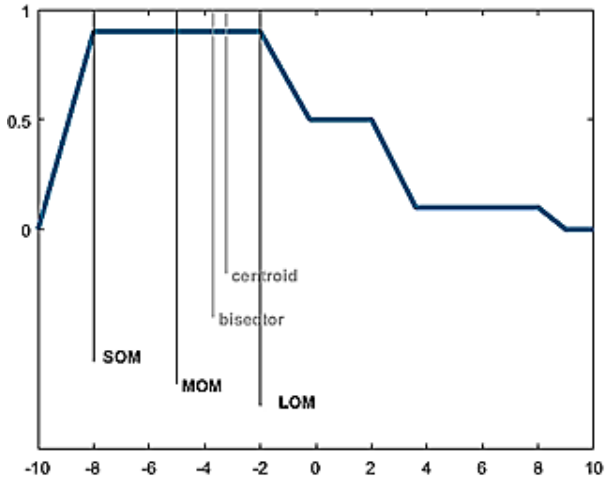


Figure 3.5 Méthodes de défuzzification [6].

Exemple [7]

Considérons un SIF (MAX-MIN) avec deux entrées : X et Y et une sortie Z . On se donne la base des règles suivante :

R1 : si X est A3 **OU** Y est B1 alors Z est C1

R2 : si X est A2 **ET** Y est B2 alors Z est C2

R3 : si X est A1 alors Z est C3

Dans la figure ci-dessous sont récapitulées les étapes d'inférence floue de la sortie du système si on considère que : $X = x_1$ *et* $Y = y_1$.

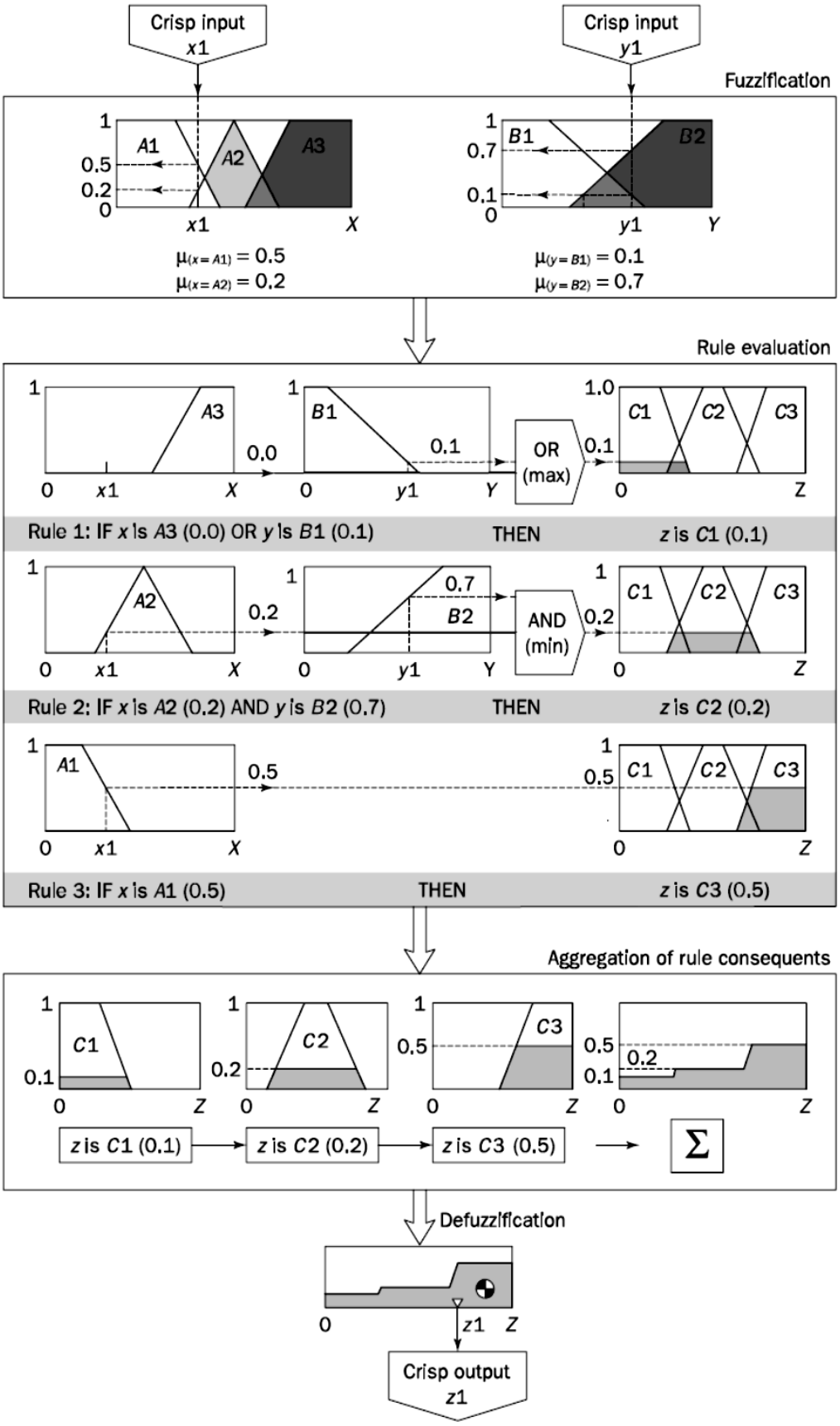


Figure 3.6 Exemple d'un SIF de type MAMDANI [7]

3.5 Modèle de Sugeno

Un scientifique a toujours besoin d'estimer (*en s'appuyant sur son savoir-faire*) si les conditions actuelles d'un système conviennent pour y appliquer des formules mathématiques déjà définies [3]. En réponse à ce besoin, vient les SIF de type Sugeno dit aussi TSK en honneur des chercheurs : *Takagi, Sugeno et Kang* ayant proposé ce modèle en 1985 [8].

Les règles d'un SIF de type TSK servent à déterminer le degré d'applicabilité d'une formule qui détermine la valeur d'une variable d'intérêt. En opposition avec le modèle linguistique de Mamdani, la sortie d'une règle TSK est numérique. Elle peut avoir la forme d'une constante, d'un polynôme, d'une équation différentielle, etc. La sortie est dépendante des variables associées à la partie antécédent [3].

Une règle dans le modèle TSK s'écrit dans le format général suivant [3]:

$$R: \quad \boldsymbol{Si}\ f(\ x_1 est\ A_1\ , x_2 est\ A_2\ , \dots , x_k est\ A_k)\ \boldsymbol{ALORS}\ y = g(x_1, x_2, .., x_k)$$

*o*ù:

- f est la fonction logique qui relie les propositions de la partie prémisse.
- $x_1, x_2, .., x_k$ sont les variables de la partie prémisse et qui apparaissent aussi dans la partie conséquence.
- $A_1\ , A_2\ , \dots , A_k$ dénotent les ensembles flous d'entrée.
- y est la variable de sortie dont la valeur doit être inférée.
- g est la fonction qui permet de calculer la valeur de la sortie y quand la partie prémisse est satisfaite.

Si la fonction g est linéaire, on aura la forme suivante:

$$R: Si\ f(\ x_1 est\ A_1\ , x_2 est\ A_2\ , \dots , x_k est\ A_k) ALORS\ y = p_0 + \ p_1 x_1 + p_2 x_2 + \cdots + p_k x_k$$

Dans un modèle *TSK d'ordre zéro*, y est une constante c'est-à-dire $p_1 = p_2 \dots = p_k = 0$, et les règles sont de la forme:

$$R: Si\ f(\ x_1 est\ A_1\ , x_2 est\ A_2\ , \dots , x_k est\ A_k)\ ALORS\ y = p_0$$

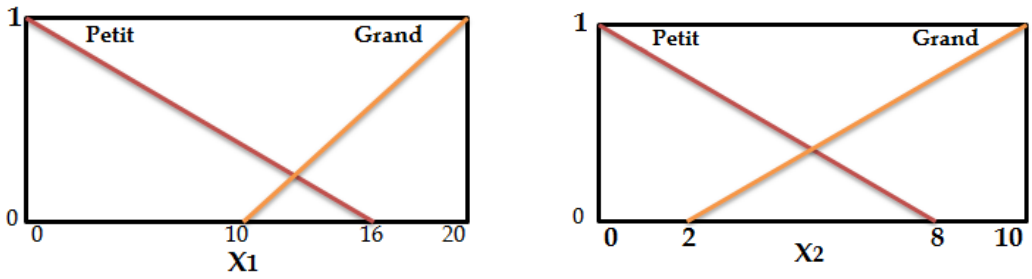
Clairement, dans un SIF de Sugeno aucune défuzzification n'est nécessaire. La sortie, y, d'un système TSK composé de r règles d'inférence se calcule en utilisant la méthode du centre de gravité comme suit :

$$\mathrm{y} = \frac{\sum_{l=1}^{\mathrm{r}} |\mathrm{y} = y_l| \times y_l}{\sum_{l=1}^{\mathrm{r}} |\mathrm{y} = y_l|}$$

- y_l dénote la conclusion obtenue par la règle l.
- $|\mathrm{y} = y_l|$ dénote le degré d'accomplissent de la règle l.

Exemple [8]

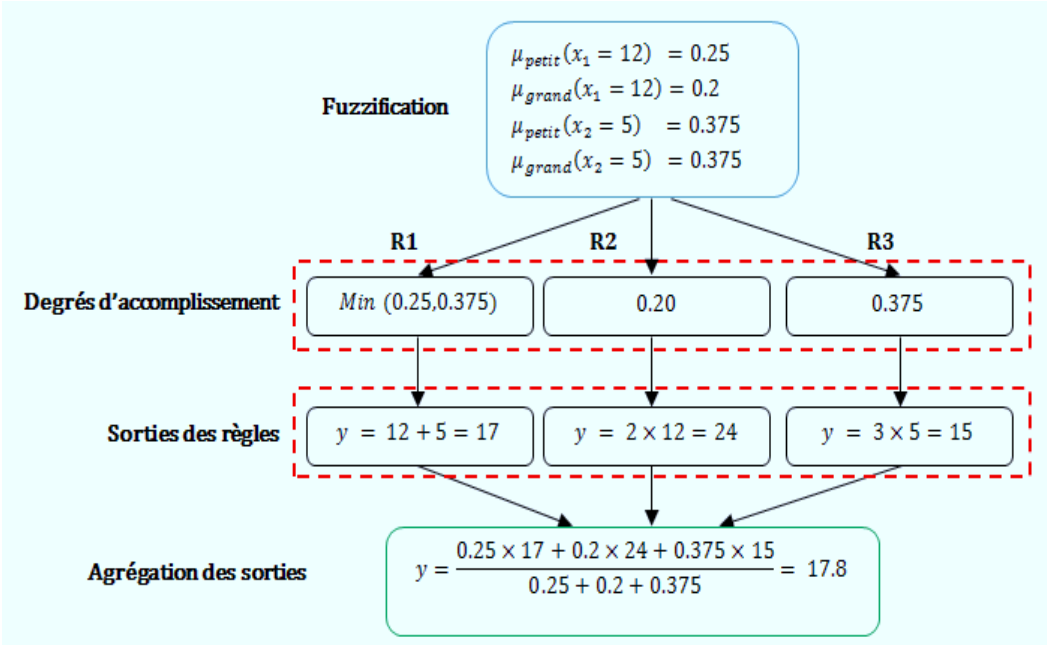
Le système possède deux entrées x_1 et x_2 et une sortie y. Les fonctions d'appartenance de x_1 et x_2 aux ensembles flous *Petit* et *Grand* sont montrées par la figure ci-dessous.



La Base des règles utilisée est la suivante :

R1:	SI x_1 est petit ET x_2 est petit	ALORS	$y = x_1 + x_2$
R2:	SI x_1 est grand	ALORS	$y = 2x_1$
R3:	SI x_2 est grand	ALORS	$y = 3x_2$

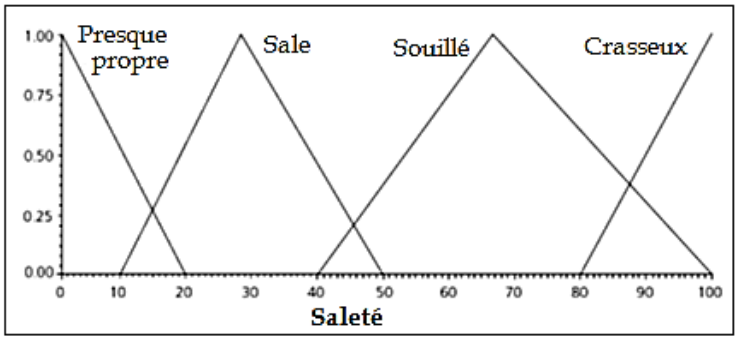
Considérons que $x_1 = 12$ et $x_2 = 5$. Récapitulons les étapes d'inférence de la valeur de la sortie y [3]:



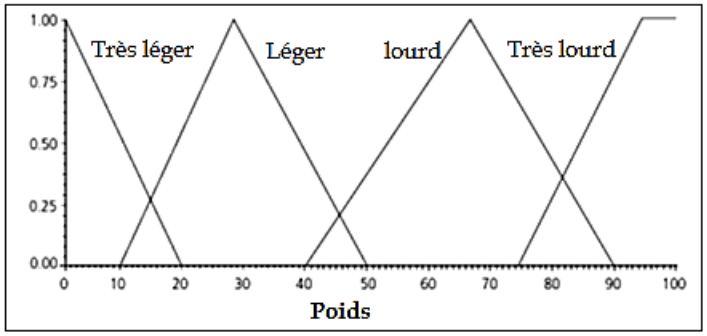
3.6 Contrôle flou d'une machine à laver

Sur le marché, il existe déjà des machines à laver utilisant des systèmes de contrôle flou (chez LG, Samsung et bien d'autres). Pour une machine à laver, les caractéristiques de la charge de linge constituent les entrées du contrôleur flou (poids, types de tissu, quantité de saleté, etc.). Selon ces données, on peut contrôler certains paramètres de lavage tel que : la quantité du détergent et de l'eau, le temps de lavage, la consommation d'électricité, etc. Le contrôle efficace de ces paramètres assure un lavage plus propre mais aussi plus économique.

Considérons, l'exemple d'une machine à laver hypothétique directement pris de la référence [9]. On suggère de contrôler la quantité du détergent en fonction du poids et de la saleté du linge.



(a)



(b)

Figure 3.7 Les ensembles flous pour l'entrée (a) Saleté (b) poids [9].

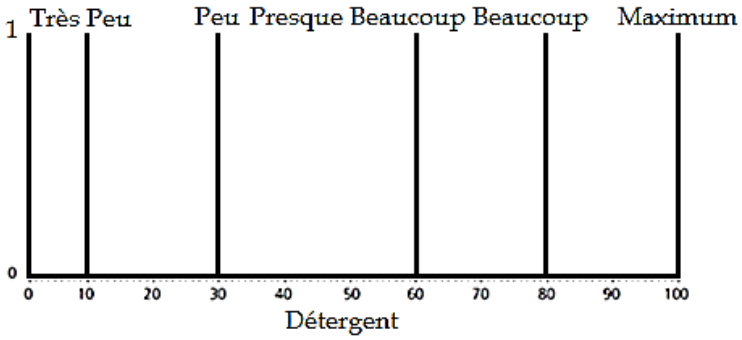


Figure 3.8 La sortie quantité détergent [9].

En examinant la sortie de ce contrôleur, on comprend qu'il s'agit d'un SIF de type Sugeno. C'est une bonne pratique de présenter l'ensemble des règles sous une forme matricielle (ici on aura 4X4=16 Règles conjonctives).

La base des règles du contrôleur de la machine à laver est comme suit [9] :

Poids / **Saleté**	***Très léger***	***Léger***	***Lourd***	***Très lourd***
Presque propre	Très peu	Peu	Presque beaucoup	Presque beaucoup
Sale	Peu	Peu	Presque beaucoup	Beaucoup
Souillé	Presque beaucoup	Presque beaucoup	Beaucoup	Maximum
Crasseux	Beaucoup	Presque beaucoup	Beaucoup	Maximum

On donne ici un exemple de lecture d'une règle à partir de la matrice précédente:

> Si **Saleté** est **presque propre** ET **poids** et **très léger** alors **détergent** = **très peu** (*exactement c'est une quantité de 10 unités de détergent*)

3.7 Contrôleur flou pour la navigation d'un robot mobile de type voiture

Afin de réaliser la commande intelligente de la navigation autonome d'un robot de type voiture, les auteurs dans la référence [10] ont proposé un régulateur flou. Ce dernier possède deux entrées : l'erreur de position (E_Pos) et l'erreur angulaire (E_Ang). Les sorties du régulateur sont la vitesse de translation et l'angle de braquage tous les deux nécessaires pour atteindre une position désirée.

Figure 3.9 Le robot voiture RobuCar utilisé dans les expérimentations [10].

Les entrées E_Pos et E_Ang numériques sont calculées comme suit [10]:

$$E_{pos} = \sqrt{D_x^2 + D_y^2} \qquad E_{Ang} = \theta_m - \theta$$

Tel que :

$$\begin{cases} D_x = X_b - X_m \\ D_y = Y_b - Y_m \\ \theta = \tan^{-1}(D_y, D_x) \end{cases}$$

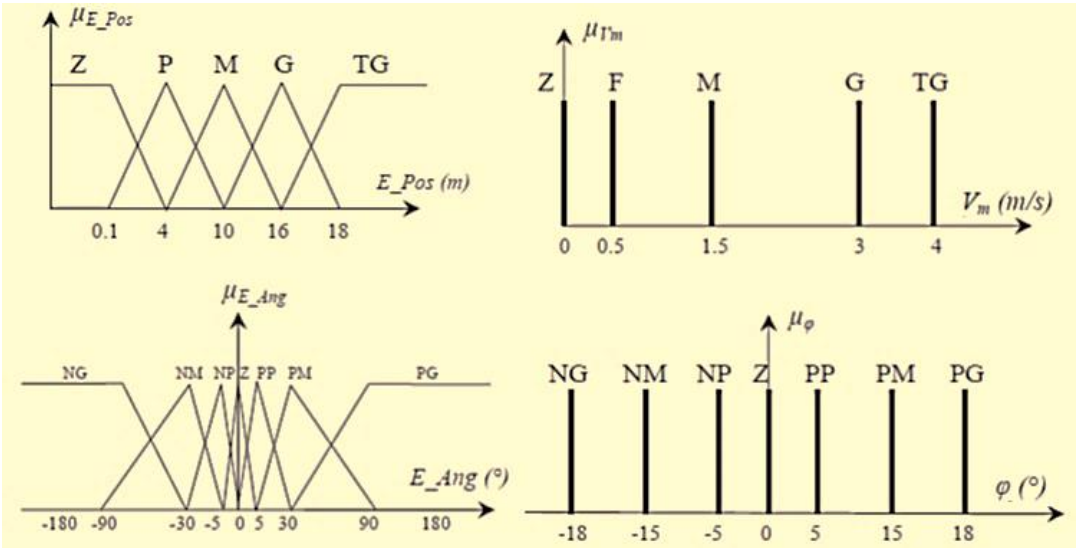
Sachant que :

- (X_m, Y_m, θ_m) : position et orientation du robot.
- (X_b, Y_b, θ_b) : position et orientation du but.
- V_m : vitesse de translation du robot
- φ : angle de braquage du robot

Les termes utilisés dans la description des différentes variables linguistiques sont :

Z : Zéro	P : Petite	PP : Positive Petite
M : Moyenne	G : Grande	PG : Positive Grande
NG : Négative Grande	F : Faible	
NP : Négative Petite	TG : Très grande	
PM : Positive Moyenne	NM : Négative Moyenne	

Les figures ci-dessous représentent les ensembles flous utilisés pour les entrées/sorties.



Voici un exemple d'une règle qui régit la navigation du robot :

> Si *l'erreur de position* est *petite* ET l'*erreur d'angle* est *nulle* (c'est-à-dire que le but est tout droit mais à une petite distance) Alors le robot doit avancer avec une *vitesse faible sans changer de direction*.

Dans l'ensemble, 70 règles sont proposées pour réguler aussi précisément que possible la navigation du robot. Le choix des règles reflète le savoir-faire d'un expert. Néanmoins, la définition des fonctions d'appartenance reste une tâche difficile (*elle se fait par expérimentation*).

La base des règles complète est donnée par la matrice suivante [10]:

Erreur de Positin (*E_Pos*)			Erreur Angulaire (*E_Ang*)						
			NG	NM	NP	Z	PP	PM	PG
Z		φ	*PM*	*PP*	*Z*	*Z*	*Z*	*NP*	*NM*
		V_m	*Z*	*Z*	*Z*	*Z*	*Z*	*Z*	*Z*
P		φ	*PG*	*PG*	*PM*	*Z*	*NM*	*NG*	*NG*
		V_m	*F*	*F*	*F*	*F*	*F*	*F*	*F*
M		φ	*PM*	*PM*	*PP*	*Z*	*NM*	*NG*	*NG*
		V_m	*F*	*F*	*M*	*M*	*M*	*F*	*F*
G		φ	*PM*	*PP*	*PP*	*Z*	*NP*	*NP*	*NM*
		V_m	*F*	*M*	*G*	*G*	*G*	*M*	*F*
TG		φ	*PM*	*PM*	*PP*	*Z*	*NP*	*NM*	*NM*
		V_m	*F*	*M*	*G*	*TG*	*G*	*M*	*F*

3.8 Conclusion

A la fin de ce chapitre, l'étudiant sera capable d'implémenter des agents intelligents qui utilisent des SIFs de type MAMDANI et/ou Sugeno. Ces SIFs présentent l'avantage d'être simples et efficaces. Ceci explique leur prolifération dans les applications industrielles.

La logique floue, comme déjà vue dans le chapitre, est une extension de la logique classique qui permet d'attribuer un degré de vérité à un énoncé. D'autres logiques non classiques qui ont une grande importance en IA seront présentées dans le chapitre suivant : il s'agit des logiques modales.

Travaux Pratiques

Sujet 1

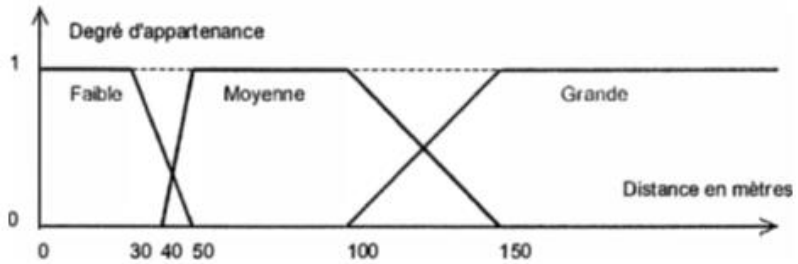
L'objet de ce TP est la conception et ensuite l'implémentation du programme agent qui utilise un SIF pour contrôler un ventilateur de maison. Le programme agent ajuste la vitesse du ventilateur en fonction de la Température et de l'Humidité de l'air dans la maison.

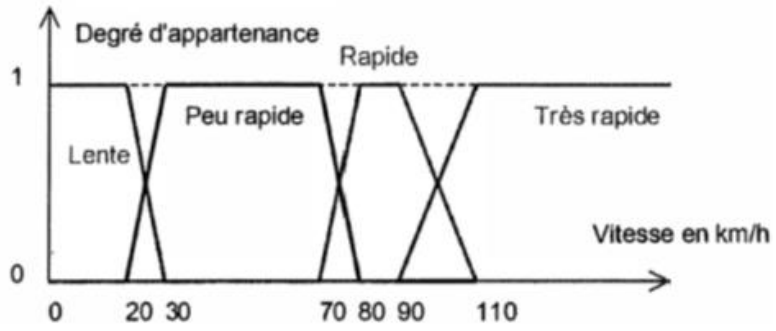
Pour réaliser ce TP :

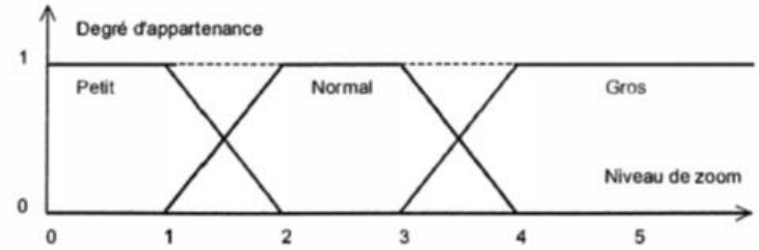
- ✓ Définir les différents ensembles flous nécessaires.
- ✓ Définir une base des règles adéquate.
- ✓ Dans un langage de votre choix : Implémenter le SIF correspondant (Mamdani ou Sugeno ?)

Sujet 2

Réaliser, en langage C, un contrôler flou du zoom GPS d'une voiture. Le zoom se définit en fonction de la distance au prochain changement de direction et de la vitesse de la voiture. Pour réaliser ce TP, on donne les différents ensembles flous nécessaires ainsi que la base des règles à utiliser [11]:







Distance → / Vitesse ↓	Faible	Moyenne	Grande
Lente	Normal	Petit	Petit
Peu rapide	Normal	Normal	Petit
Rapide	Gros	Normal	Petit
Très rapide	Gros	Gros	Petit

Références du chapitre

[1] L. Zadeh, "*Is there a need for fuzzy logic?*", Information Sciences, vol. 178, no. 13, pp. 2751-2779, 2008.

[2] L. Zadeh, "*Fuzzy sets*", Information and Control, vol. 8, no. 3, pp. 338-353, 1965.

[3] S. Chettibi, "*Intelligence Computationnelle pour les problèmes de Routage adaptatif efficace en énergie et multicritère dans les Réseaux Mobiles Ad-hoc*", Thèse de doctorat en sciences (Chapitre 3) , Université de Constantine 2, 2015.

[4] B. Bouchon-Meunier, C. Marsala and J. Pomerol, *Logique floue, principes, aide à la décision*. Paris: Hermès Science publications, 2003.

[5] E.H. Mamdani and S. Assilian, "*An experiment in linguistic synthesis with a fuzzy logic controller,*" *International Journal of Man-Machine Studies*, Vol. 7, No. 1, pp. 1-13, 1975.

[6] https://www.mathworks.com/help/fuzzy/examples/defuzzification-methods.html

[7] M. Negnevitsky, "*Artificial intelligence: A Guide to Intelligent Systems*". Harlow: Addison Wesley,2010.

[8] T. Takagi and M. Sugeno, "*Fuzzy identification of systems and its applications to modeling and control*", IEEE Transactions on Systems, Man, and Cybernetics, vol. -15, no. 1, pp. 116-132, 1985.

[9] A. Ibrahim, "*Fuzzy logic for embedded systems applications*". Amsterdam: Newnes, 2004.

[10] N. Ouadah, 0. Azouaoui, M. Hamerlain, ''*Implémentation d'un contrôleur flou pour la navigation d'un robot mobile de type voiture''*. Troisième Congres francophone, MAJECSTIC 2005, 16-18 Novembre 2005, Rennes (France).

[11] V. Mathivet, ''*L'Intelligence Artificielle pour les développeurs''*, ISBN: 978-2-7460-9215-0, Editions ENI, Décembre 2014.

CHAPITRE 4

Logiques Modales

4.1 Introduction

La formalisation logique des systèmes mono-agent ou multi-agent se fait dans deux perspectives différentes, à savoir [1] : i) l'implémentation du raisonnement automatique dans des agents cognitifs ; ii) la spécification et la vérification du comportement des agents (*qui ne sont pas forcément cognitifs*) situés dans des environnements dynamiques. De par leurs expressivités, les logiques modales sont souvent utilisées dans ce contexte.

En linguistique, une ***modalité*** a pour effet de modifier le sens d'un énoncé. En logique, une modalité sert à qualifier le ***mode de vérité*** des assertions. On peut citer par exemple:

- les *modalités aléthiques* comme : « *il est nécessaire que* », « *il est possible que* »,
- les *modalités épistémiques* comme: « *un tel sait que* »,
- les *modalités doxastiques* comme: « *un tel croit que* »,
- les *modalités temporelles* permettant d'exprimer *le futur et le passé*.

Ces modalités sont celles étudiées dans : la *logique propositionnelle modale*, la *logique des savoir et des croyances* et les *logiques temporelles*, respectivement.

La *logique modale* peut être définie comme [2] « *l'étude du raisonnement sur des modalités et de l'inférence à partir de prémisses modales qu'une certaine conclusion modale est valide* ».

4.2 Logique modale propositionnelle

La *logique modale propositionnelle* étend la logique propositionnelle avec les opérateurs modaux universel et existentiel. L'interprétation généralement attribuée à ces opérateurs est celle de ***nécéssité/possibilité***. On utilise la notation suivante :

$\Box Q$: « *il est nécessaire que Q* ». $\Diamond Q$: « *il est possible que Q*».

Au fait, ces deux opérateurs sont duaux (*Cf. Figure 4.1*) :

$\Box Q \leftrightarrow \neg \Diamond \neg Q$ $\Diamond Q \leftrightarrow \neg \Box \neg Q$

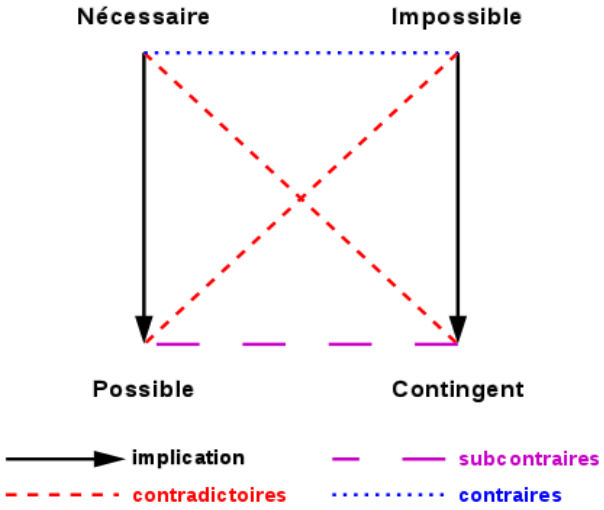


Figure 4.1 Le carré modal[1].

Notons que d'autres interprétations à part la nécessité et la possibilité peuvent être associées à ces opérateurs à condition que ces interprétations satisfassent la relation de dualité. En voici des exemples :

Interprétations de $\Box Q$	**Interprétations de $\Diamond Q$**
Il sera toujours vrai que Q	Il sera parfois vrai que Q
Il est su que Q	L'inverse de Q n'est pas su
Il est cru que Q	L'inverse de Q n'est pas cru
Il faut que Q	Il est permis que Q
Toute exécution du programme produit le résultat Q	Il y a une exécution du programme qui produit le résultat Q

Tableau 4.1 Interprétations des opérateurs modaux [3].

Soulignons que les deux dernières interprétations dans le tableau précédent donnent lieu à la ***logique déontique*** et la ***logique dynamique***, respectivent (*cf. Section 4.2.6*) .

Exemple

Soit la proposition atomique « Q : *la porte est fermée* ». Donner la signification des formules suivantes en utilisant l'interprétation « savoir » pour un certain agent situé dans une pièce :

[1] https://fr.wikipedia.org/wiki/Logique_modale

$\Box Q$	l'agent sait que la porte est fermée.
$\Box\neg Q$	l'agent sait que la porte n'est pas fermée.
$\Diamond Q$	l'agent ne sait pas si la porte n'est pas fermée.
$\Diamond \neg Q$	l'agent ne sait pas si la porte est fermée.
$\Box \Box Q$	l'agent sait qu'il sait que la porte est fermée.
$\Box\neg\Box Q$	l'agent sait qu'il ne sait pas que la porte est fermée.

4.2.1 Syntaxe de la logique modale propositionnelle

Soit $\mathcal{L}_p$ le langage propositionnel et Φ l'ensemble des propositions atomiques. Le langage modal $\mathcal{L}_M$ est défini comme suit [1]:

- Vrai et Faux ∈ $\mathcal{L}_M$
- Si $\psi \in \Phi$ Alors $\psi \in \mathcal{L}_M$
- Si p, q ∈ $\mathcal{L}_M$ Alors $p \wedge q, \neg p, p \vee q, p \rightarrow q, p \leftrightarrow q \in \mathcal{L}_M$
- Si $p \in \mathcal{L}_{\boldsymbol{p}}$ Alors $\Box p, \quad \Diamond p \in \mathcal{L}_M$

4.2.2 Mondes possibles et structure de Kripke

L'idée est de considérer que chaque agent peut envisager un certain nombre de mondes concevables. On dit alors qu'un agent *sait une proposition* si elle est vraie dans tous les mondes qu'il imagine. En effet, la vérité n'est pas absolue, elle peut dépendre de la « réalité » autrement dit du « monde » dans lequel on se trouve.

On appelle interprétation en terme de mondes possibles **un modèle de Kripke**[2] $\mathcal{M} =< \boldsymbol{W}, \boldsymbol{L}, \boldsymbol{R} >$ tel que [1]:

- W : un ensemble de mondes ou situations possibles.
- L : $\boldsymbol{W} \rightarrow \boldsymbol{2}^{\Phi}$: une fonction qui renvoie l'ensemble des propositions atomiques satisfaites dans un monde ($\boldsymbol{2}^{\Phi}$dénote l'ensemble des parties de Φ).
- $R \sqsubseteq W \times W$: est une relation d'accessibilité binaire. $(\omega, \omega') \in R$ indique que le monde ω' est accessible depuis le monde ω.

Récapitulons, pour chaque interprétation de $\Box Q$, le sens qu'on peut attribuer à la relation d'accessibilité :

Interprétations de $\Box \boldsymbol{Q}$	**Sens de R($\boldsymbol{\omega}, \boldsymbol{\omega}'$)**
Il est nécessairement vrai que Q	ω' est un monde possible selon ω
Il sera toujours vrai que Q	ω' est un monde future de ω
L'agent A sait que Q	ω' peut être le monde actuel selon les connaissances de A dans ω
L'agent A croit que Q	ω' peut être le monde actuel selon les croyances de A dans ω
Il est obligatoire que Q	ω' est un monde acceptable selon l'information dans ω
Après toute exécution du programme P, Q arrive	ω' est un état résultant possible après l'éxécution de P dans ω.

Tableau 4.2 Interprétations de la relation d'accessibilité [4].

[2] En l'honneur de *Saul Kripke* Philosophe et logicien américain, auteur de l'article révolutionnaire intitulé « *A Completenes Theorem for Modal Logic* » portant sur la sémantique de la logique modale.

4.2.2.1 Sémantique des mondes possibles

Dans ce qui suit, w dénote le monde par rapport auquel on évalue une formule. La valeur de vérité des formules modales est définie inductivement comme suit [1] :

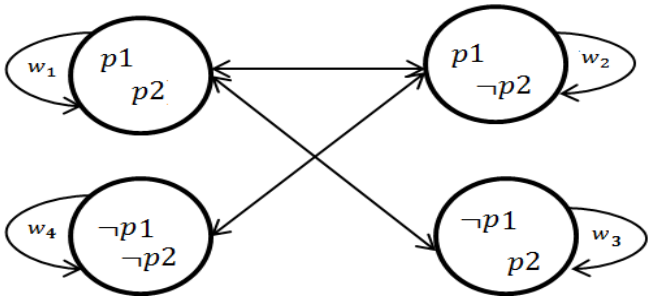
$\mathcal{M}, w \vDash \psi \; SSI \; \psi \in L(w) \; tel \; que \; \psi \in \Phi$
$\mathcal{M}, w \vDash p \wedge q \quad SSI \quad \mathcal{M}, w \vDash p \; et \; \mathcal{M}, w \vDash q$
$\mathcal{M}, w \vDash \neg p \quad SSI \quad \mathcal{M}, w \nvDash p$
$\mathcal{M}, w \vDash \Diamond p \quad SSI \; (\exists w` : R(w, w`) \wedge \mathcal{M}, w' \vDash p)$
$\mathcal{M}, w \vDash \Box p \quad SSI \; (\forall w` : R(w, w`) \rightarrow \mathcal{M}, w' \vDash p)$

On peut donner une interpretation informelle pour les deux dernières règles comme suit :

- La formule $\Diamond p$ est satisfaite dans un monde possible ω ssi la formule p est satisfaite dans au moins un monde possible accessible à partir de ω
- La formule $\Box p$ est satisfaite dans un monde possible ω ssi la formule p est satisfaite dans tous les mondes possibles accessibles à partir de ω.

Exemple

Soit la structure de Kripke $\mathcal{M}$ ci-dessous.



On peut constater par exemple que :

$\mathcal{M}, w_1 \vDash p1$
$\mathcal{M}, w_2 \vDash p1$
$\mathcal{M}, w_1 \vDash \Box(p1 \vee p2)$
$\mathcal{M}, w_2 \nvDash \Box (p1 \vee p2)$

4.2.2.2 Propriétés algébriques des relations d'accessibilité

Dans un modèle $\mathcal{M} = < W, L, R >$, la relation d'accessibilité peut être [4] :

- Réflexive SSI $(\forall \omega : (\omega, \omega) \in R)$
- Sérielle SSI $(\forall \omega : (\exists \omega' : (\omega, \omega') \in R))$
- Transitive SSI $(\forall \omega , \omega', \omega'' : (\omega, \omega') \in R \wedge (\omega', \omega'') \in R \rightarrow (\omega, \omega'') \in R)$
- Symétrique SSI $(\forall \omega , \omega' : (\omega, \omega') \in R \rightarrow (\omega', \omega) \in R)$
- Euclidienne SSI $(\forall \omega , \omega', \omega'' : (\omega, \omega') \in R \wedge (\omega, \omega'') \in R \rightarrow (\omega', \omega'') \in R)$

Et on parle dans ce cas de modèle réflexif, sériel, transitif, symétrique et euclidien, respectivement [4].

Notons que la relation d'accessibilité définie dans *l'exemple précédent*, est : réflexive, sérielle, non transitive, symétrique mais non euclidienne.

4.3 Axiomes de la logique modale

Les principaux axiomes de la logique modale sont [1,4] :

- A**xiome K** (ou axiome de distribution), en l'honneur de Saul Kripke, dont le schéma est le suivant:

$$\Box (p \rightarrow q) \rightarrow (\Box p \rightarrow \Box q)$$

Si un agent croit/sait une implication entre deux termes, alors s'il croit/sait le premier, il croit/sait aussi le second. Cet axiome est souvent considéré comme l'élément principal des ***logiques des savoirs et des croyances***. Il est valide dans tout modèle de Kripke quelque soit la relation d'accessibilité.

- **Axiome T** ou encore axiome de connaissance , possède le schéma suivant:

$$\Box p \rightarrow p$$

Cet axiome est valide dans la classe des modèles caractérisés par une relation d'accessibilité réfléxive. Il exprime que l'agent sait seulement ce qui est actuellement vrai. Autrement, une connaissance est une information vraie.

- **Axiome 4** ou encore axiome d'introspection[3] positive, exprimé comme suit:

$$\Box p \rightarrow \Box \Box p$$

C'est l'axiome caractéristique de la classe des modèles avec une relation d'accessibilité transitive. Il exprime qu'un agent est conscient de ce qu'il connait.

- **Axiome 5** ou encore axiome d'introspection négative:

$$\Diamond p \rightarrow \Box \Diamond p$$

C'est l'axiome caractéristique de la classe des modèles avec une relation d'accessibilité Euclidienne. Cet axiome exprime qu'un agent est conscient de ce qu'il ignore.

- **Axiome D** ou encore axiome de non contradiction :

$$\Box p \rightarrow \Diamond p$$

C'est l'axiome caractéristique de la classe des modèles avec une relation d'accessibilité Sérielle. Cet axiome exprime que si un agent croit/sait quelque chose, alors il ne croit/sait pas son contraire.

4.4 Systèmes de la logique modale

Le système formel K contient uniquement [5]:

1) Les tautologies du calcul propositionnel
2) L'axiome K : $\Box(p \rightarrow q) \rightarrow (\Box p \rightarrow \Box q)$
3) La formule duale : $\Diamond p \leftrightarrow \neg\Box\neg p$

La logique modale minimale est construite à base du système formel K clos par [5]:

- Le modus ponens:

$$\frac{p \qquad p \rightarrow q}{q}$$

- Ainsi que la règle de nécessité suivante:

$$\frac{p}{\Box p}$$

[3] Observation individuelle de la conscience elle-même.

En effet, plusieurs systèmes logiques peuvent être obtenus en conservant différents axiomes comme le montre la figure ci-dessous.

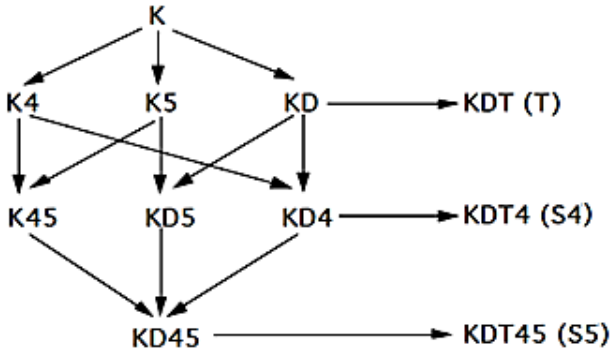


Figure 4.2 Systèmes de la logique modale [6].

Le choix d'un système modal dépendra du concept à modéliser [3] :

- Si on veut caractériser la connaissance d'un agent intelligent ayant une parfaite capacité d'introspection logique sur *ce qu'il connait*, on choisira le système modale **S4**. Si, en plus, l'agent est conscient de ce qu'il ne connait pas on choisira le système modal **S5.**

Les systèmes S4 et S5 sont la base des logiques de savoir.

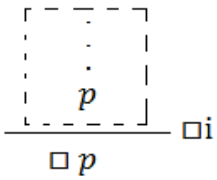
- Si on désire caractériser les croyances d'un agent idéalement rationnel (i.e. un agent dont certaines croyances peuvent se révéler erronées mais qui cependant possède une parfaite faculté d'introspection logique sur ce qu'il croit et ne croit pas), on choisira le système K45.

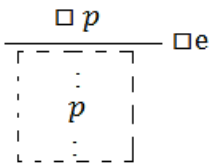
Les systèmes K45 et KD45 sont la base des logiques de croyance.

Exemples [4]

1) Montrer par ***déduction naturelle*** que : $\vdash_K (\Box p \wedge \Box q) \rightarrow \Box(p \wedge q)$

En plus des règles de déduction naturelle de la logique propositionnelle, on ajoute les deux règles suivantes:





Tel que :

$\Box i$: indique l'introduction de l'opérateur modal □

$\Box e$: indique l'élimination de l'opérateur modal □

Notons qu'il n'ya pas de règles explicite pour l'opérateur de possibilité : il est écrit dans les preuves sous la forme équivalente $\neg\Box\neg p$.

Dans une preuve :

- Le fait d'entrer dans une *boîte en ligne continue* désigne qu'on est en train d'établir une *hypothèse.*
- Le fait d'entrer dans une *boîte en pointillés* indique qu'on est en train de raisonner dans un *monde possible* arbitraire.

1	$\Box p \wedge \Box q$	Hypothèse
2	$\Box p$	$\wedge e$ 1
3	$\Box q$	$\wedge e$ 1
4	p	$\Box$ e 2
5	q	$\Box$ e 3
6	$p \wedge q$	$\wedge i$ 4,5
7	$\Box(p \wedge q)$	$\Box i$ 4-6

8 $\Box p \wedge \Box q \rightarrow \Box(p \wedge q)$ $\rightarrow i$ 1-7

Rappelons les règles de déduction de la logique propositionnelle utilisées dans cet exemple et qui servent respectivement à l'introduction/élimination de la conjonction et à l'introdution de l'implication :

$$\frac{p \quad q}{p \wedge q} \wedge i \qquad \frac{p \wedge q}{p} \wedge e \qquad \frac{\boxed{\begin{array}{c} p \\ \vdots \\ q \end{array}}}{p \rightarrow q} \rightarrow i$$

2) Montrer par déduction naturelle que :

$$\vdash_{KT45} (p \rightarrow \Box \Diamond p)$$

En plus des règles déjà vues, exploitons les deux règles suivantes qui servent à l'introduction/élimination de la négation :

$$\frac{\boxed{\begin{array}{c} p \\ \vdots \\ \bot \end{array}}}{\neg p} \neg i \qquad \frac{p \quad \neg p}{\bot} \neg e$$

1	p	Hypothèse
2	$\Box\neg p$	Hypothèse
3	$\neg p$	Axiome *T sur* 2
4	$\bot$	$\neg e$ 1, 3
5	$\neg\Box\neg p$	$\neg$i 2-4
6	$\Box\neg\Box\neg p$	Axiome 5 sur 5

7 $p \rightarrow \Box\neg\Box\neg p$ $\rightarrow i$ 1-6

4.5 Raisonnement sur les connaissances dans un système multi-agent

Au sein d'un même ***système multi-agent,*** les différents agents ont différentes connaissances sur le monde. Dans certaines applications, un agent doit raisonner non seulement sur ses propres connaissances mais aussi sur les connaissances des autres agents. Contrairement à l'être humain, un agent artificiel peut suivre aisément une imbrication d'énoncés du genre: « *Papa ne sait pas si le Voisin sait que Papa sait que le Voisin sait que le facteur est passé ce matin* ».

Néanmoins, la formalisation de tels énoncés (*et donc le raisonnement sur ces derniers*) requière la généralisation des logiques modales déjà vues au contexte multi-agent. La logique $KT45^n$ [4] constitue un exemple dont le soin est laissé aux étudiants curieux.

4.6 Logique dynamique et Logique déontique

La logique dynamique peut être vue comme la ***logique modale de l'action***. La logique dynamique propositionnelle des programmes réguliers est la variante la plus populaire de cette logique. La sémantique de cette dernière est donnée respectivement à un model composé d'un ensemble d'états (ou mondes possibles) où la transition d'un état à un autre se fait en exécutant un programme atomique (*une action que l'agent exécute directement*) [1].

La logique déontique, quant à elle, sert à modéliser l'obligation et la permission et par dualité le facultatif et l'interdiction. Etant un ensemble d'actions possibles, quelle action doit choisir l'agent si son fonctionnement est régi par une norme ou une loi ? La réponse n'est pas du tout évidente, d'ailleurs plusieurs paradoxes sont soulevés contre la logique déontique (*consulter la référence [5] pour en avoir une idée*).

4.7 Logiques temporelles

En logique temporelle, les mondes possibles représentent les états du monde évalués à différents moments. La relation d'accessibilité peut prendre le sens de postériorité *(ce qui permet d'exprimer le futur)* ou d'antériorité *(ce qui permet d'exprimer le passé)*:

Lecture de ω R ω'	**Lecture □ Q**	**Lecture ◊ Q**
t est postérieur à s	à tous les instants futurs Q	à au moins un instant futur Q
t est antérieur à s	à tous les instants passés Q	à au moins un instant passé Q

Tableau 4.3 Interprétations de la relation d'accessibilité et des opérateurs modaux en logique Temporelle [3].

Selon la représentation du temps qu'on adopte : linéaire ou arborescente, on distingue deux familles de logique temporelle, à savoir : la logique LTL et la logique CTL. Ces deux logiques font l'objet des sous-sections suivantes.

4.7.1 Logique LTL

La logique LTL (*Linear Temporal Logic*) a été introduite la première fois par *Amir Pnueli* en 1977 dans le cadre de vérification des programmes informatiques[4]. L'interprétation d'une formule LTL est définie en termes de chemins (séquences d'état) : où à chaque état est associé un seul successeur ou autrement « *un seul futur possible*».

4.7.1.1 Syntaxe de la logique LTL

La logique LTL étend la logique propositionnelle avec des opérateurs temporels. Soit $\mathcal{L}_p$ le langage propositionnel , le langage LTL $\mathcal{L}_L$ est défini comme suit [1]:

- Les règles de $\mathcal{L}_p$ s'appliquent sur $\mathcal{L}_L$
- Si $p, q \in \mathcal{L}_L$ alors $p\boldsymbol{U}q, \boldsymbol{X}p,\ \boldsymbol{F}p, \boldsymbol{G}p \in \mathcal{L}_L$

Formule	Lecture	Sémantique
$\boldsymbol{pUq}$	$p\ Until\ q$	p est toujours vrai jusqu'à un moment où q soit vrai
$\boldsymbol{Xp}$	$Next\ p$	p sera vérifié au prochain moment
Fp	Finally p	p sera vérifié plus tard dans le future
Gp	Globally p	p est toujours vérifié.

Tableau 4.4 Signification des opérateurs de la logique LTL.

Remarques

$$Fp \equiv \text{True} \cup \text{p} \qquad\qquad Gp \equiv \neg\text{F}\neg\text{p}$$

Exemple [7]

« Un ascenseur qui se déplace vers le haut et se trouve en deuxième étage ne change jamais sa direction quand il a des clients qui souhaitent se rendre au cinquième étage ».

On veut modéliser cette règle de fonctionnement en LTL. Pour ce faire, on utilise les propositions atomiques suivantes:

- Etage_2 : l'ascenseur se trouve au deuxième étage.
- Etage_5 : l'ascenseur se trouve au cinquième étage.
- Haut : la direction de l'ascenseur est vers le haut.
- BApp_5 : bouton 5 appuyé.

La formule correspondante est: $\mathbf{G}(\mathbf{Etage\_2} \wedge \mathbf{Haut} \wedge \mathbf{BApp\_5} \rightarrow (\mathbf{Haut\ U\ Etage\_5}))$

4.7.1.2 Sémantique de la logique LTL

Soient l'ensemble Φ des propositions atomiques et la structure de Kripke $\mathcal{M} =< W, L, \mathrm{w}_0, R >$ où $\mathrm{w}_0 \in W$ désigne l'état initial. Un chemin infinie dans la structure $\mathcal{M}$ est une suite $\pi = \mathrm{w}_0\mathrm{w}_1 \ldots \mathrm{w}_t \ldots$ tel que :$\forall\, t \in \mathbb{N}$: $(\mathrm{w}_t, \mathrm{w}_{t+1}) \in R$. En plus, considérons les notations suivantes :

$\pi(i)$: dénote le i[eme] état dans le chemin π.

π_i : dénote la queue du chemin π qui commence à la position i.

[4] *LTL a été introduite dans l'article intitulé «The temporal logic of programs» pour la vérification des programmes séquentiels et parallèles.*

La relation de satisfaction entre un chemin π et une formule LTL se définit comme suit [8]:

$\mathcal{M}, \pi \vDash \psi \quad SSI \; p \in L(\pi(0)) \; tel \; que \; \psi \in \Phi$

$\mathcal{M}, \pi \vDash \neg p \quad SSI \quad \mathcal{M}, \pi \nvDash p$

$\mathcal{M}, \pi \vDash \quad p \wedge q \quad SSI \quad \mathcal{M}, \pi \vDash p \; et \quad \mathcal{M}, \pi \vDash q$

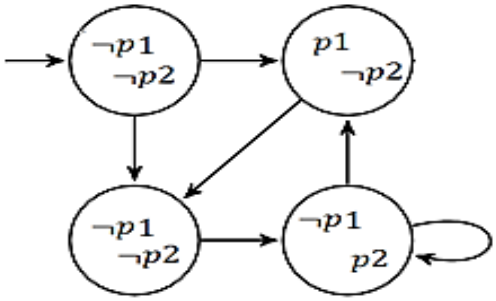
$\mathcal{M}, \pi \vDash Xp \quad SSI \; (\mathcal{M}, \pi_1 \vDash p)$

$\mathcal{M}, \pi \vDash p \; U \; q \quad SSI \; (\exists \; k \in \mathbb{N} : \mathcal{M}, \pi_k \vDash q \; et \; \forall \; 0 \leq i < k \; \mathcal{M}, \pi_i \vDash p)$

La sémantique est ici définie par rapport à un chemin quelconque. Par extension, une structure de Kripke $\mathcal{M}$ d'origine w_0 satisfait une formule si et seulement si elle est vérifiée pour tout chemin dans $\mathcal{M}$ d'origine w_0.

Exemple

Soit la structure de Kripke ci-dessous. Donner des exemples de formules LTL satisfaites dans cette dernière.



4.7.2 Logique CTL

Edmund M. Clarke et Allen Emerson ont inventé la logique CTL (*Computation Tree Temporal Logic*) et le model checking en 1981. En CTL, le temps est vu comme une structure d'arbre où pour chaque état du système plusieurs futurs sont possibles selon l'action qui sera effectuée.

4.7.2.1 Syntaxe de la logique CTL

La logique CTL étend la logique LTL par les quantificateurs de chemin ***A et E***. Soit p une formule LTL :

- ***Ap***: signifie que tous les chemins (exécutions) partant de l'état courant satisfont p.
- ***Ep***: signifie qu'il existe au moins un chemin partant de l'état courant qui satisfait p.

Notons que les deux opérateurs sont duaux : $\boldsymbol{Ep \equiv \neg A \neg P}$

Soit $\mathcal{L}_L$ le langage LTL, le langage CTL $\mathcal{L}_C$ est défini comme suit :

- Les règles de $\mathcal{L}_L$ s'appliquent sur $\mathcal{L}_C$
- Si $p, q \in \mathcal{L}_C$ alors $AXp, EXp, AFp, EFp, AGp, EGp, A(p \; Uq), E(pUq) \in \mathcal{L}_C$

4.7.2.2 Sémantique de la logique CTL

Soit $\mathcal{M}$ une structure de Kripke. La sémantique des formules CTL se définit par rapport à un sommet de $\mathcal{M}$, en considérant les chemins ayant pour origine ce sommet [8] :

$\mathcal{M}, s \vDash \psi \quad SSI \quad p \in L(s)\ tel\ que\ \psi \in \Phi$

$\mathcal{M}, s \vDash \neg p \quad SSI \quad \mathcal{M}, s \nvDash p$

$\mathcal{M}, s \vDash \quad p \wedge q \quad SSI \quad \mathcal{M}, s \vDash p\ et\ \mathcal{M}, s \vDash q$

$\mathcal{M}, s \vDash EXp \quad SSI \quad (\exists\ \pi$ tel que $\pi(0) = s$ et $\mathcal{M}, \pi_1 \vDash p)$

$\mathcal{M}, s \vDash AXp \quad SSI \quad (\forall\ \pi$ tel que $\pi(0) = s$ et $\mathcal{M}, \pi_1 \vDash p)$

$\mathcal{M}, s \vDash \quad E(p\ U\ q) \quad SSI \quad (\exists\ \pi$ tel que $\pi(0) = s,$

$\exists\ k \in \mathbb{N} : M, \pi_k \vDash q\ et \forall\ 0 \leq i < k\ \mathcal{M}, \pi_i \vDash p$

$\mathcal{M}, s \vDash \quad A(p\ U\ q) \quad SSI \quad (si\ \pi$ est un chemin tel que $\pi(0) = s\ alors$

$\exists\ k \in \mathbb{N} :\ M, \pi_k \vDash q\ et \forall\ 0 \leq i < k\ \mathcal{M}, \pi_i \vDash p$

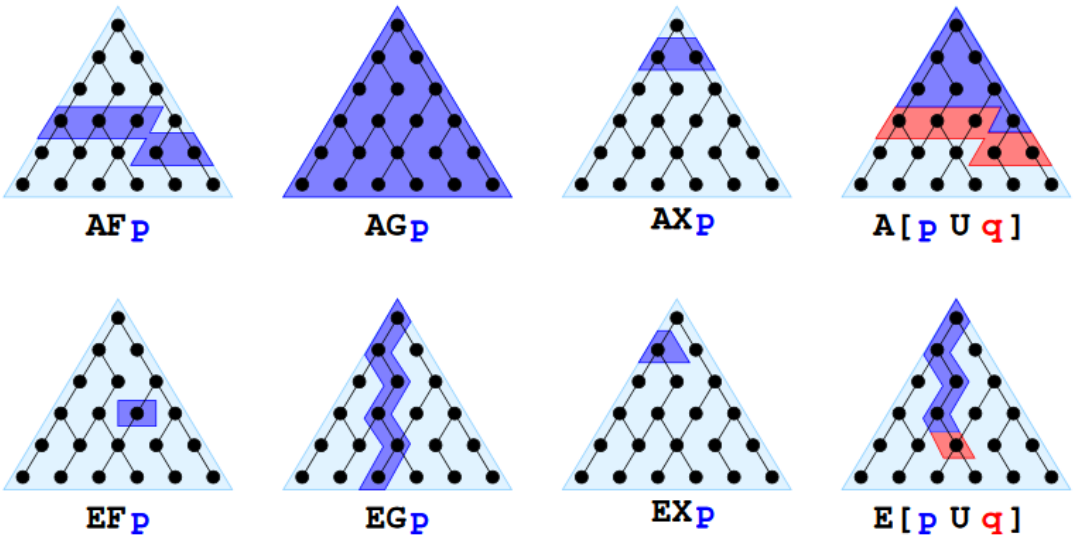
Et par extention :

$\mathcal{M} \vDash p\ SSI\ \mathcal{M}, w_0 \vDash p$

On peut donner une description informelle aux quatre dernières règles comme suit :

- La formule ***EXp*** est satisfaite dans un état si la formule ***p*** est satisfaite dans au moins un de ses successeurs.
- La formule ***AXp*** est satisfaite dans un état si la formule ***p*** est satisfaite dans tous ses successeurs.
- La formule **E(pUq)** est satisfaite dans un état s'il existe au moins un chemin issu de cet état tel que p est toujours satisfaite jusqu'à un état qui vérifie q.
- La formule **A(pUq)** est satisfaite dans un état si sur chaque chemin issu de cet état p est toujours satisfaite jusqu'à un état qui vérifie q.

Pour mieux comprendre la sémantique des différents opérateurs, on donne la représentation graphique ci-dessous [5].



4.7.3 Vérification de modèle (ou Model Checking)

L'objectif du model checking est la vérification des propriétés de bon fonctionnement sur un système réactif. (*Au fait, un agent intelligent peut être vu comme un système réactif*).

Un algorithme de model checking prend en entrée : i) une abstraction du système réactif étudié (*présentée via un modèle de Kripke* $\mathcal{M}$) et ii) une spécification du

[5] http://www.inf.unibz.it/~artale/FM/slide4.pdf

comportement désiré (*exprimée pas une formule écrite en logique temporelle* φ). L'algorithme vérifie si l'abstraction satisfait ou non la formule $\mathcal{M} \vDash \varphi$? (*c.-à-d. si elle est un modèle pour cette formule d'où le terme anglais model checking*). Si la réponse est négative : un contre-exemple est retourné [9].
Souvent, on cherche à vérifier les propriétés suivantes:

- **Invariance** : tous les états du système satisfont une bonne propriété.
- **Accessibilité** : une certaine situation peut être atteinte.
- **Sûreté :** quelque chose de mauvais n'arrive jamais.
- **Vivacité :** sous certaines conditions, un certain évènement souhaitable finira par se produire.

Pratiquement, SPIN[6] et NuSMV[7] sont les *model-checker* les plus utilisés. Mise à part la différence dans les langages de modélisation utilisés par les deux logiciels, SPIN supporte seulement des propriétés écrites en LTL tandis que NuSMV supporte à la fois LTL et CTL [11].

Exemple (LTL) [10]

Considérons un feu de circulation routière avec les trois états : vert, jaune et rouge. Spécifions en LTL les propriétés désirables suivantes:

- « Le feu devient finalement vert » :

$$F\ vert$$

- « Une fois rouge, le feu ne peut pas devenir vert directement » :

$$G(rouge \rightarrow \neg X vert)$$

- « Une fois rouge, le feu devient vert après avoir été jaune pour un certain temps » :

$$G(rouge \rightarrow X(\ jaune\ \wedge X(jaune\ U\ vert))$$

Exemple (CTL)

Imaginons un distributeur de boisson qui fonctionne comme suit. La machine se trouve dans un état initial d'attente. Pour avoir du thé, le client doit insérer une pièce de monnaie. Pour avoir du café, il faut insérer deux pièces de monnaie (une après l'autre). Ensuite, le bouton OK doit être appuyé. Enfin, la machine doit revenir sur son état initial. Considérons $\Phi = \{a, b, c, d, e, f, g, h\}$ tel que :

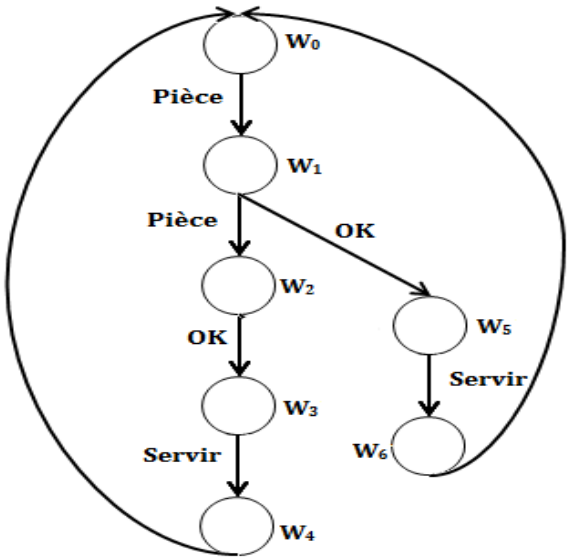
✓ a : Machine en état d'attente	✓ e: Café en cours de préparation
✓ b : 1 pièce de monnaie déjà insérée	✓ f : Thé en cours de préparation
✓ c : 2 pièces de monnaie déjà insérées	✓ g : Café servi
✓ d : Bouton ok déjà appuyé	✓ h : Thé servi

1. On veut spécifier quelques propriétés de bon fonctionnement pour ce distributeur de boisson :

[6] http://spinroot.com/spin/whatispin.html
[7] http://nusmv.fbk.eu/

- La machine peut préparer du café.
- La machine finit toujours par préparer du thé si une pièce de monnaie est insérée et qu'ensuite le bouton Ok a été appuyé.
- La machine ne serve jamais du thé si l'utilisateur a inséré deux pièces de monnaie.
- Une fois une boisson est servie, la machine doit se remettre directement en attente.

2. Soit la structure de Kripke $\mathcal{M}$ ci-dessous qui modélise le fonctionnement de notre machine (des actions sont utilisées pour étiqueter les transitions).



- Définir la fonction L.
- Quelles propriétés algébriques vérifie la relation d'accessibilité ?
- Soit le chemin infini : $\pi = w_0\ w_1\ w_3$ .... Vérifier si (justifier votre réponse):
  $\mathcal{M}, \pi \vDash X\ \neg c$
  $\mathcal{M}, \pi_2 \vDash X\ (c \wedge d)$
  $\mathcal{M}, \pi_3 \vDash X\ a$

4.8 Conclusion

Le thème général de ce chapitre est les *logiques modales*. Nous avons mis en avant la diversité des logiques sur lesquelles on peut s'appuyer dans la formalisation d'une telle composante ou autre au sein d'un système mono ou multi agent. Nous incitons l'étudiant à imaginer comment mêler ces différentes logiques ensemble ?

Dans un système intelligent, on parle aussi d'une composante très importante qui est l'interface homme-machine. Le développement de cette dernière repose, entre autres, sur les outils du TALN (*Traitement Automatique du Langage Naturel*). Le chapitre suivant constitue une introduction à cette branche fondamentale de l'intelligence artificielle.

Recueil d'exercices
Logiques des propositions, des prédicats, modales, CTL et LTL.

Les exercices 1 à 4 sont inspirés de la référence [12].

Exercice 1 *(Modélisation à l'aide de la logique des propositions)*

Un système de sécurité d'une automobile peut être caractérisé par les *propositions atomiques* suivantes:

Proposition	***Fait représenté***
M	Moteur en état de marche
C	La caméra en état de marche
A	Marche arrière enclenchée
Pi (i=1,..,4)	La porte i est ouverte
H	La pression d'huile est suffisante
R	La voiture roule
AP	L'alerte porte est en marche
AH	L'alerte pression d'huile est en marche

Q1. Donner les formules représentant les axiomes suivants (dits *invariants du système*):

- **F1 :** La caméra se met en marche lorsque la marche arrière est enclenchée.
- **F2** : Si le moteur est en marche et la pression d'huile est insuffisante, l'alerte pression d'huile doit être enclenchée.
- **F3** : L'alerte porte doit être enclenchée lorsque la voiture roule et une des portes est ouverte.

Q2. Soit les assignations des propositions A1, A2, A3 et A4 :

	A1	**A2**	**A3**	**A4**
M	V	V	V	V
C	F	**V**	**V**	**V**
A	F	F	F	**V**
P1	F	F	F	F
P2	F	F	F	F
P3	F	F	**V**	F
P4	F	F	F	F
H	V	V	V	V
R	V	V	V	**F**
AP	F	F	F	F
AH	F	F	F	F

Etant donnée l'assignation **A1,** le système est-il dans un état correct ?

Q3. Pour que le système reste dans un état correct, tout évènement qui modifie la valeur de vérité d'une ou de plusieurs propositions n'est admissible que s'il fait passer les invariants du système d'un modèle à un autre. Alors, les évènements suivants sont-ils interdits (*considérons que le système se trouve initialement dans A1*) ?

- Activer la caméra.
- Activer la caméra et ouvrir la porte 3.
- Enclencher la marche arrière, activer la caméra et s'arrêter de rouler.

Exercice 2 *(La logique des propositions a un pouvoir d'expression limité)*
Considérons un domaine de connaissance composé de 3 objets, 2 propriétés, 1 relation binaire symétrique.

- Combien de variables propositionnelles a-t-on besoin pour représenter nos connaissances sur le domaine en question?
- Représenter en LP les énoncés suivants:
  - Il y a au moins un objet pour lequel la propriété 2 est vraie.
  - Il y a au moins deux objets pour lesquels la propriété 2 est vraie et qui sont en relation.
- Refaire les mêmes questions pour: 10 objets, 15 propriétés et 15 relations. Commenter.
- Maintenant, on ne sait pas combien il y a d'objets dans le système. Représenter l'énoncé suivant: Tout objet possède soit la propriété 1 soit la propriété 2.

Exercice 3 *(Rappel de la Syntaxe de la logique des prédicats)*
Soit les prédicats suivants :

P(x) : x est un professeur
H(x) : x est un humain
S(x) : x est un étudiant
L(x) : x est un cours
B(x,y) : x assiste le cours y

Exprimer en logique des prédicats les énoncés suivants:

- Tous les étudiants sont des humains
- Aucun étudiant n'est professeur
- Quelques humains sont des professeurs
- Certains étudiants assistent à tous les cours
- Tous les étudiants n'assistent pas à tous les cours

Exercice 4 *(Sémantique, domaine de connaissance, fonction d'interprétation)*
Soit le langage composé des symboles:

- **Constantes**: a, b, c, d, k1, k2, k3, k4
- **Prédicats:** P (unaire), E (unaire), C (unaire), H (binaire), S(unaire), T (binaire)
- **Variables:** x, y, z.

On interprète ce langage dans le domaine suivant:

Personne = {Ali, Lila ,Omar,Dina}
Etudiant = {Ali, Lila,Omar}
Cours = {Logique, Algèbre,Analyse,statistique}

On définit une interprétation sur ce domaine comme suit:

$\mathcal{I}(a) = Ali$	$\mathcal{I}(k1) = Logique$	$\mathcal{I}(P) = Personne, \mathcal{I}(E) = Etudiant$
$\mathcal{I}(b) = Lila$	$\mathcal{I}(k2) = Algèbre$	$\mathcal{I}(C) = Cours,$
$\mathcal{I}(c) = Omar$	$\mathcal{I}(k3) = Analyse$	$\mathcal{I}(S) = Enseignant$
$\mathcal{I}(d) = Dina$	$\mathcal{I}(k4) = Statistique$	$\mathcal{I}(T) = Enseigne$

$\mathcal{I}(H) = Inscription$={(Ali,Logique),(Lila,statistique), (Omar,Logique)}

Evaluer les formules suivantes (Vrai ou faux) selon l'interprétation $\mathcal{I}$:

- $C(K3)$
- $C(a)$
- $H(c, K1)$
- $\exists\, x\, H(x, k2)$
- $\forall x \exists\, y\, H(x, y)$

Exercice 5 *(Logique Modale)*

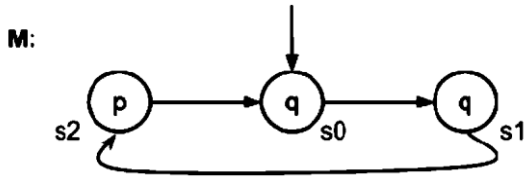
Considérer un modèle de Kripke $\mathcal{M} =< W, L, R >$:

1. Quand $\mathcal{M}, w \vDash p \Rightarrow q$?
2. Vérifier que :
   « Si $\mathcal{M}$ est un modèle quelconque alors l'***axiome K est valide*** (formellement : $\mathcal{M}, w \vDash \Box(p \Rightarrow q) \Rightarrow (\Box p \Rightarrow \Box q)$) »
3. Vérifier que :
   « Si $\mathcal{M}$ est muni d'une relation d'accessibilité réflexive alors ***l'axiome T est valide*** (formellement : $\mathcal{M}, w \vDash (\Box p \Rightarrow p)$) »

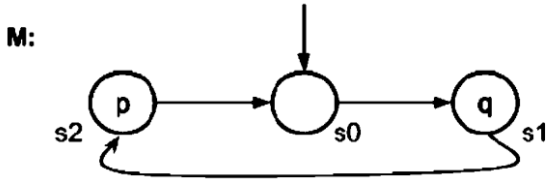
Exercice 6 *(Logiques Temporelles)*

1. Exprimer en LTL et en CTL les énoncés suivants :
   - p n'est jamais vérifié.
   - À chaque fois que p arrive, q doit éventuellement arriver.

2. Soit $\mathcal{M}$ le modèle (système de transition) de la figure. Vérifier si $\mathcal{M} \vDash G(p \vee q)$ et si $\mathcal{M} \vDash Gp \vee Gq$



3. Vérifier si $\mathcal{M} \vDash F(p \wedge q)$ et si $\mathcal{M} \vDash Fp \wedge Fq$ pour le modèle suivant :



Exercice 7 [10]

Considérons un canal de communication unidirectionnel entre deux processus S (émetteur) et R (récepteur). S est équipé par un tampon de sortie **S.out** et R est équipé par un tampon d'entrée **R.in**. Quand S envoie un message, il le met dans S.out. Le récepteur R reçoit un message en le supprimant de R.in.

Soient les propositions atomiques:

X : un message m est dans S.out **Y**: un message m est dans R.in

Formaliser en LTL, les spécifications de fonctionnement suivantes qui doivent être toujours vérifiées:

- Si un message est dans S.out alors m sera finalement reçu.
- Un message m reste dans S.out jusqu'à sa réception.
- Un message ne peut être en même temps dans S.out et R.in.
- Si l'émetteur envoie un message **m** suivi d'un autre message **p** alors le récepteur doit recevoir **m** avant **p** (préservation de l'ordre d'envoie).

Références du chapitre

[1] M. P. Singh, A. S. Rao, and M. P. Georgeff. *Formal methods in DAI: Logic based representation and reasoning*. In G. Weis, editor, *Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence*, chapter 8, pages 331–376.

[2] R. Mastop *Modal Logic for Artificial Intelligence***:** notes de cours, 2012. Accessible à *http://www.phil.uu.nl/~rumberg/infolai/Modal_Logic.pdf.*

[3] A. Thayse, *De la logique modale à la logique des bases de données*. Paris: Dunod, 1989. MIT Press, Cambridge, MA, 1999.

[4] M. Huth and M. Ryan, *Logic in computer science*. Cambridge: Cambridge University Press, 2000.

[5] J. Duparc, *La logique pas à pas*, Presses polytechniques et universitaires romandes. ISBN 978-2-88915-126-4, 2015.

[6] J. Ferber, *Les systèmes multi-agents. Vers une intelligence collective*. Paris : InterEditions, 1995.

[7] J. Johnson, *Temporal Logic and the NuSMV Model Checker: notes de cours CS 680 Formal Methods*, Drexel university.

[8] P. Bourdil, *Contribution à la modélisation et la vérification formelle par model-checking – Symétries pour les Réseaux de Petri Temporels* : Thèse de doctorat (Chapitre 1), *INSA de Toulouse*, 2015.

[9] S. Bardin, *Introduction au Model Checking : notes de cours*, Université de Paris-Saclay, 2017.

[10] C. Baier and J. Katoen, *Principles of model checking*. Cambridge [MA]: The MIT Press, 2008.

[11] Y. Jiang and Z. Qiu, *S2N: Model Transformation from SPIN to NuSMV*. In: Donaldson A., Parker D. (eds) Model Checking Software. SPIN 2012. Lecture Notes in Computer Science, vol 7385. Springer, Berlin, Heidelberg, 2012.

[12] G. Falquet, *Fondements formels des systèmes d'information (chapitre 4 : logique propositionnelle),* Université de Genève, Faculté SES, département HEC, Centre universitaire d'informatique, 2012.

CHAPITRE 5

Introduction au Traitement Automatique du Langage Naturel

5.1 Introduction

Un *langage* est un système qui contient un ensemble de symboles, une syntaxe et une sémantique. La syntaxe permet de former, à partir des symboles, des expressions complexes dont l'interprétation est définie par la sémantique. En ce sens, les systèmes logiques, les langages de programmation et les langues naturelles sont tous des langages. Par *langage naturel* (en opposition au *langage formel*), on entend un langage parlé et écrit par des humains comme: l'Arabe, le Français, etc.

Dans ce chapitre, nous nous intéressons au *Traitement Automatique du Langage Naturel* (TALN). Le TALN regroupe l'ensemble des recherches et développements visant à modéliser et reproduire, à l'aide de machines, la capacité humaine à produire et à comprendre des énoncés linguistiques dans des buts de communication [1]. Le TALN est un champ pluridisciplinaire qui fait appel bien évidement à l'informatique et la linguistique mais aussi à la logique, les statistiques, la psychologie et les sciences cognitives.

Une langue naturelle est intrinsèquement ambiguë. Malgré cela, l'être humain est capable de désambiguïser le sens des différentes constructions linguistiques. La reproduction d'une telle faculté au sein d'une application de TALN n'est pas du tout une tâche triviale.

Le traitement de la langue se fait à différent niveaux superposés : lexical, syntaxique et sémantique. Au sein d'un même niveau, différentes voies peuvent être prises. Pour ce chapitre, qui est loin d'être exhaustif, l'étudiant est initié à l'utilisation de l'approche logique (où on fait recours notamment au langage Prolog) et aussi à l'approche statistique/probabiliste (où nous référons au modèle de Markov caché pour la résolution du problème particulier d'étiquetage morphosyntaxique). Nous nous intéressons également au calcul de mesures de similarité à base de thésaurus comme une alternative pour le traitement sémantique lexicale.

Afin de bien assimiler certaines notions théoriques présentées dans le chapitre, l'utilisation du langage Python conjointement avec la bibliothèque NLTK est recommandée. L'étudiant doit surtout apprendre à manipuler des ressources textuelles via NLTK.

5.2 Niveaux d'analyse linguistique dans une application du TALN

Le traitement de la langue permettrait à l'ordinateur d'analyser et de comprendre d'énormes quantités de données textuelles, de communiquer avec nous par écrit ou par parlé, de capturer nos mots indépendamment de l'interface d'entrée (à travers du clavier ou par reconnaissance de parole), d'analyser nos phrases, de comprendre nos propos, de répondre à nos questions voire même de mener des conversations avec nous [2]. Aujourd'hui, les applications du TALN sont multiples, on cite entre autres: la correction orthographique, la recherche et l'indexation d'information, les interfaces vocales, la reconnaissance de l'écriture, la traduction automatique, le résumé automatique, la dictée vocale, etc.

Le processus général du TALN passe par les étapes schématisées dans la figure ci-dessous. Ce processus constitue sans doute l'axe principal de ce chapitre. Mais avant d'aborder les différents traitements, soulignons que:

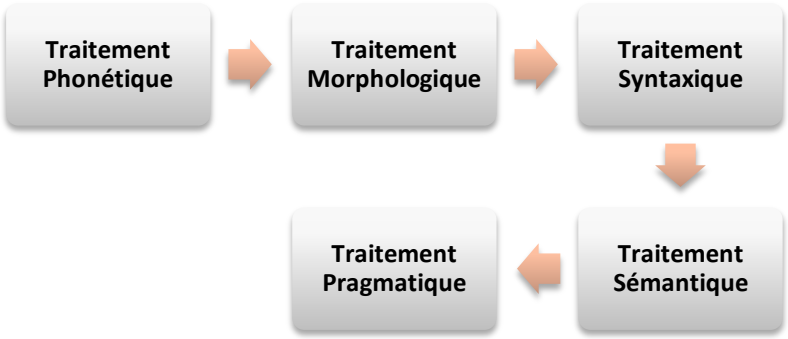


Figure 5.1 Processus général du TALN.

- Pour aller du son au sens, différentes techniques peuvent être utilisées à chaque niveau *(traitement de signal, statistiques, probabilité, logique, raisonnement automatique…)*
- Le traitement phonétique n'est effectué que lorsque l'entrée du système est vocale (*ce n'est pas d'ailleurs le cas pour toutes les applications citées ci-avant*).

- Etant donnée une phrase, l'analyse morphologique permet de déterminer les « mots » qui sont par la suite regroupés en structures par l'analyse syntaxique.
- Le traitement sémantique se fait aux niveaux lexical et syntaxique. Il s'agit dans les deux cas d'une *sémantique hors contexte*.
- Le traitement pragmatique peut être vu comme un traitement *sémantique en contexte*. Dans cette phase, on fait lier le sens de la phrase à son contexte spécifique pour aboutir à sa signification réelle. La pragmatique a pour objet l'étude des *actes du langage* où on se préoccupe des concepts importants suivants [3]:
  - *l'implicature* qui se réfère à ce qui est signifié implicitement par le locuteur. Par exemple, la phrase: « Pouvez-vous fermer la porte ? » n'est pas une question, mais plutôt une forme de politesse via laquelle on demande à quelqu'un de fermer la porte.
  - *les présuppositions* qui sont les faits qui doivent être vrais afin qu'une phrase peut être évaluée comme vraie ou fausse. Par exemple, si on dit «Ahmed est guéri» cela présuppose qu'Ahmed était malade.
  - *l'étude de discours* c'est-à-dire la manière dont l'homme connecte des phrases pour transmettre l'information.

Par manque de temps, nous n'abordons pas dans ce cours les travaux menés en matière de pragmatique. La discussion de la sémantique syntaxique nous renvoie à l'approche logique dans la représentation/interprétation du sens : une compétence déjà acquise par les étudiants au cours des chapitres précédents. Néanmoins, nous allons un peu plus profondément en termes de sémantique lexicale qui est naturellement à la base de la sémantique syntaxique.

5.3 Modèles et Algorithmes

En TALN, on doit représenter les diverses connaissances linguistiques nécessaires à chaque niveau d'analyse (*phonétique, morphologique, syntaxique, sémantique et pragmatique*). Dans ce but, généralement on fait recours aux automates à états finis, aux grammaires formelles, à la logique du premier ordre mais aussi à des modèles probabilistes pour remédier aux problèmes d'ambiguïté [4].

Une fois la phase de modélisation est accomplie, on est amené typiquement à résoudre des problèmes de recherche dans un espace d'états. Pour cette fin, des algorithmes bien connus comme la recherche en profondeur d'abord *(pour des tâches non probabilistes)* et de programmation dynamique *(pour des tâches probabilistes*) peuvent être appliqués [4].

Dans ce chapitre, nous donnons des exemples sur l'utilisation du langage Prolog, qui est un langage basé sur la logique des prédicats, en TALN. Prolog est exploité pour implémenter des automates et des grammaires et puis mener des recherches pour répondre à des problèmes de reconnaissance et de génération souvent rencontrés aux niveaux lexical et syntaxique. Aussi, le Modèle de Markov Caché (MMC) est présenté conjointement avec l'algorithme de Viterbi (*qui implémente le principe de la programmation dynamique*) dans le contexte particulier d'étiquetage

morphosyntaxique. Enfin, nous abordons le calcul de similarité lexicale par thésaurus.

5.4 Quelques notions préliminaires en TALN

Dans cette section sont définies les notions de n-grammes, de corpus et de thésaurus : des notions auxquelles nous faisons recours dans le reste du chapitre.

5.4.1 N-grammes

Dans un modèle probabiliste du langage, on fait l'hypothèse que chaque mot ne dépend que des (n-1) mots qui le précèdent. On parle souvent de [5] :

- Trigramme (n = 3) : un mot dépend des deux mots qui le précèdent.
- Bigramme (n = 2) : un mot dépend du mot qui le précède.
- Unigramme (n = 1) : un mot ne dépend d'aucun mot qui le précède.

L'apprentissage de la distribution de probabilité des n-grammes se fait à partir d'un corpus.

5.4.2 Corpus

Un *corpus* est [6] : « une collection de données langagières qui sont sélectionnées et organisées selon des critères linguistiques et extralinguistiques explicites pour servir d'échantillon d'emplois déterminés d'une langue. Pratiquement, un corpus doit exister sous un format électronique où il est encodé de manière standardisée et homogène pour permettre des extractions non limitées à l'avance».

Le *corpus de Brown* est un exemple de corpus annoté (avec 87 étiquettes morphosyntaxiques). C'est une collection de 1 million de mots provenant de 500 textes de différents genres (journaux, romans, académiques, etc.), rassemblés à l'Université Brown en 1963 [4].

5.4.3 Thésaurus

Un *thésaurus*, aussi appelée ontologie en linguistique, est une organisation hiérarchique de termes censés être représentatifs d'un certain domaine. Les termes sont liés entre eux notamment par la relation spécifique/générique (*par exemple : un bateau est un véhicule*), la relation de synonymie (*par exemple : un navire est synonyme de bateau*) et la relation «terme associé» (*par exemple : navigation est un terme associé à bateau*). Les thésaurus sont souvent utilisés pour l'indexation de documents et la recherche de ressources documentaires.

Le *thésaurus WordNet* est une base lexicale de la langue anglaise qui couvre plus de 150 000 mots qui varient entre noms, verbes, adjectifs et adverbes. Sa richesse mais aussi sa disponibilité en ligne[1] (*peut-être même téléchargé sous forme de faits Prolog*) sont les clés de son grand succès. WordNet arrange les mots ainsi que les significations de mots dans une matrice lexicale, tel que [2]:

[1] https://wordnet.princeton.edu/

- Une ligne horizontale définit un ensemble de mots synonymes dits *synset* dans le vocabulaire WordNet.
- Une colonne montre les différentes significations que peut avoir un mot.

5.5 Programmation Logique avec Prolog

Un algorithme consiste d'une partie logique qui décrit les connaissances nécessaires (*le quoi ou le sens de l'algorithme*) et d'une partie contrôle qui décrit comment exploiter ces connaissances pour trouver la solution désirée. Dans le paradigme impératif traditionnel, on décrit explicitement les étapes de résolution d'un problème. En revanche, en programmation déclarative y compris en Programmation Logique (PL), on s'intéresse plutôt au sens du programme.

Les caractéristiques d'un langage de programmation logique peuvent être résumées dans les points suivants :

- Un langage de programmation logique est destiné à la programmation symbolique où les données traitées sont des informations non numériques.
- Dans un programme logique, on décrit des objets, des propriétés et des relations ce qui donne l'aspect déclaratif.
- Etant donné un programme logique, qui n'est rien autre qu'un ensemble de faits et de règles, un langage de programmation logique utilise le processus d'inférence logique pour répondre aux requêtes.

5.5.1 Rappel des notions de la logique des prédicats utilisées en PL

a) Forme clausale [7] : toutes les formules de la logique des prédicats peuvent s'écrire dans le format suivant

$$A_1 \wedge A_2 \wedge \dots A_m \rightarrow B_1 \vee B_2 \dots \vee B_n$$

Informellement « *si tous les A sont vrais alors au moins un B est vrai* ». Voici un exemple d'une formule écrite en forme clausale :

$$\text{Père (Ali, Omar)} \wedge \text{Mère (Lila, Omar)} \wedge \text{GrandPère(Ahmed, Omar)} \rightarrow \text{Père(Ahmed, Ali)} \vee \text{Père(Ahmed, Lila)}$$

Exemple (adapté de la référence [8])

Représenter les énoncés en langage naturel suivants dans la forme clausale :

« *Tout Américain qui vend des armes à des pays hostiles est un criminel (1). Le pays B possède des missiles (2), B est un ennemi de l'Amérique(3). Toutes ses missiles lui ont été vendu par colonel C (4) qui est américain (5)*»

(1) $Américain(x) \wedge Arme(y) \wedge Vend(x, y, z) \wedge Hostile(z) \rightarrow Criminel(z)$

(2) $\exists\, x\ Possède\ (B, x) \wedge Missile(x)$, par instanciation existentielle où on introduit une constante M1 on obtient:
$Possède\ (B, M1)$
$Missile(M1)$

(3) $Ennemi\ (B, Amérique)$
Il faut aussi ajouter (pour faire le lien entre Hostile et Ennemi):
$Ennemi\ (x, Amérique) \rightarrow Hostile(x)$

(4) $Missile(x) \wedge Possède(B, x) \rightarrow Vend(C, x, B)$

(5) $Américain(C)$

(6) $Missile(x) \rightarrow Arme(x)$

b) *Clauses de Horn [7] :* sont les clauses ayant au maximum un littéral positif. On distingue trois formes de clause de Horn :
   - Règle : composée d'un littéral positif avec au moins un littéral négatif.
   - Fait : avec seulement un littéral positif.
   - Objectif ou requête : avec seulement des littéraux négatifs.

c) *Résolution / unification [7] :* la résolution s'applique sur des formules écrites au format clausale. Elle permet de déduire de nouvelles connaissances en appliquant le mécanisme suivant:
   - Appliquer la conjonction entre les parties gauches et la disjonction entre les parties droites.
   - Supprimer les éléments qui apparaissent dans les deux côtés.

Exemple

Dans cet exemple : a, b, c sont des constantes.

$$parent(a,b) \rightarrow père(a,b) \vee mère(a,b)$$
$$père(a,b) \wedge père(b,c) \rightarrow grandpère(a,c)$$

La résolution mène à :

$$parent(a,b) \wedge père(b,c) \rightarrow grandpère(a,c) \vee mère(a,b)$$

En présence de variables, il faut trouver des valeurs pour ces variables permettant de réussir le processus d'appariement : on parle d'unification.

Exemple

$père(x) \rightarrow homme(x)$

$père(Ali)$

Il faut substituer x par Ali :

$père(Ali) \rightarrow homme(Ali)$

$père(Ali)$

La résolution de ces deux clauses donne :~~$père(Ali)$~~ $\rightarrow homme\ (Ali) \vee$ ~~$père(Ali)$~~

Ce qui coïncide avec le fait : $homme\ (Ali)$

5.5.2 Langage Prolog

Prolog[2] (*PROgrammation LOGique*) est sans doute le langage logique le plus connu. Il a été inventé à Marseille en 1972 par *Alain Colmerauer*. Prolog est un langage de l'intelligence artificielle conçu à l'origine pour des applications du traitement automatique du langage naturel et des grammaires formelles. Nous résumons les principales propriétés du langage Prolog à travers plusieurs exemples [7]:

a) Prolog utilise les clauses de Horn (les faits, les règles et les requêtes).
   Voici un programme ou une base de connaissance Prolog qui contient deux faits et une règle:

```
pluvieux (batna).
froid (batna).
neigeux (X) :- pluvieux(X), froid(X).
```

   $Soient\ les\ requêtes$:

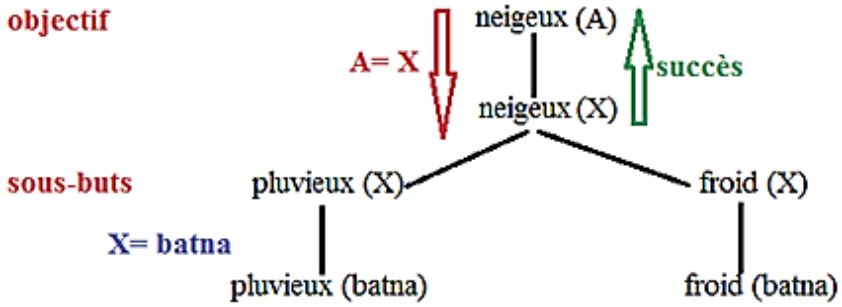
[2] Les captures d'écrans figurant dans cette section sont obtenues par l'environnement *swi-prolog* téléchargeable à *http://www.swi-prolog.org/*

$?-froid\ (batna).$
$?-neigeux\ (batna).$
$?-neigeux(A).$

Pour répondre à une requête, la base de connaissance est parcourue de la première clause à la dernière.

b) Prolog utilise le *chainage en arrière* et effectue une *recherche en profondeur d'abord* (preuve sous-but par sous-but):

Essayons de répondre à : $?-neigeux\ (A)$. Pour ce faire, construisons un arbre :



Examinons de près la trace d'exécution Prolog correspondante (où call désigne *le début d'une vérification*, Exit indique que *le but est vérifié*, le symbole = dénote l'*unification*) :

```
[trace]  ?- neigeux(A).
   Call: (8) neigeux(_582) ? creep
   Call: (9) pluvieux(_582) ? creep
   Exit: (9) pluvieux(batna) ? creep
   Call: (9) froid(batna) ? creep
   Exit: (9) froid(batna) ? creep
   Exit: (8) neigeux(batna) ? creep
A = batna.
```

c) Prolog effectue *un retour arrière*, s'il échoue dans la vérification d'un sous but :

Ajoutons en tête de la BC de l'exemple précédent le fait pluvieux (jijel) et exécutant la même requête précédente. On obtiendra la trace suivante (où *Fail* désigne *l'echec d'une vérification*, *Redo* indique *un retour arrière et l'essai d'une nouvelle instanciation*) :

```
[trace]  ?- neigeux(A).
   Call: (8) neigeux(_702) ? creep
   Call: (9) pluvieux(_702) ? creep
   Exit: (9) pluvieux(jijel) ? creep
   Call: (9) froid(jijel) ? creep
   Fail: (9) froid(jijel) ? creep
   Redo: (9) pluvieux(_702) ? creep
   Exit: (9) pluvieux(batna) ? creep
   Call: (9) froid(batna) ? creep
   Exit: (9) froid(batna) ? creep
   Exit: (8) neigeux(batna) ? creep
A = batna.
```

d) L'ordre des clauses dans la BC et celui des termes dans les règles affectent l'efficacité du processus d'inférence :
   - Dans l'exemple précédent, si on n'a pas ajouté pluvieux (jijel) en tête de la BC on aurait pu aboutir à une réponse plus rapidement (en évitant le retour arrière).
   - Etant donnée la requête : $?-hostile\ (X)$. Il vaut mieux utiliser la règle : $hostile(X):-ennemi(X, amérique), pays\ (X)$. que d'utiliser : $hostile\ (X):-pays(X), ennemi(X, amérique)$. puisqu'il y a logiquement moins d'ennemis que de pays (*on minimise le nombre des retours arrières qu'on peut y avoir*).

e) Prolog utilise l'opérateur Cut, symbolisé par (!), pour contrôler la recherche. Voici un exemple comparatif (*sans et avec Cut*) :

```
chemin (X,Y) :- arc (X,Z), arc (Z,Y).
arc(1,6).
arc(1,8).
arc(6,7).
arc(6,1).
arc(8,3).
arc(8,1).
```

Soit la requête ?- chemin (1,V).

```
[trace]  ?- chemin(1,v).
   Call: (8) chemin(1, v) ? creep
   Call: (9) arc(1, _1134) ? creep
   Exit: (9) arc(1, 6) ? creep
   Call: (9) arc(6, v) ? creep
   Fail: (9) arc(6, v) ? creep
   Redo: (9) arc(1, _1134) ? creep
   Exit: (9) arc(1, 8) ? creep
   Call: (9) arc(8, v) ? creep
   Fail: (9) arc(8, v) ? creep
   Fail: (8) chemin(1, v) ? creep
false.
```

```
chemin (X,Y) :- arc (X,Z), !, arc (Z,Y).
arc(1,6).
arc(1,8).
arc(6,7).
arc(6,1).
arc(8,3).
arc(8,1).
```

Soit la requête ?- chemin (1,V).

```
[trace]  ?- chemin(1,v).
   Call: (8) chemin(1, v) ? creep
   Call: (9) arc(1, _1132) ? creep
   Exit: (9) arc(1, 6) ? creep
   Call: (9) arc(6, v) ? creep
   Fail: (9) arc(6, v) ? creep
   Fail: (8) chemin(1, v) ? creep
false.
```

ici Z est unifié à 6 seulement, le retour arrière est bloqué par le cut (!)

f) Prolog, tout comme les langages fonctionnels, utilise intensivement les listes :

On donne ici l'exemple du prédicat append qui permet de concaténer des listes. Il est prédéfini dans Prolog:

$append([], X, X).$
$append([X|Y], Z, [X|W]):-append(Y, Z, W).$

g) Prolog est basé sur l'hypothèse du monde fermé : en réalité, Prolog est un système de type True|Fail et non pas True| False (Prolog retourne false quand il échoue à démontrer le True à partir de sa base de connaissance).

h) La négation dans Prolog prend le sens de l'échec. Voici un exemple : la requête: $?-not\ (ennemi\ (X, algérie))$. est équivalente à : $\neg\ \exists X\ (ennemi(X, algérie))$.On ne peut pas exprimer qu'on cherche des pays qui ne sont pas des ennemis de l'Algérie , c'est-à-dire $\exists X\ (\neg ennemi(X, algérie))$?

5.6 Chaines de Markov

Une chaine ou *processus de Markov* est une séquence de variables aléatoires $\{X_1, X_2, \dots, X_T\}$ qui prennent leurs valeurs dans l'espace d'états : $S = \{q_1, q_2, \dots, q_N\}$. Un processus stochastique est dit *markovien* s'il vérifie les deux propriétés suivantes [2]:

- *Historique limité* : l'état courant dépend d'un nombre constant d'états précédents. Par exemple, on parle de processus du premier ordre lorsque:

$$P(X_t = q_j | X_1, X_2, \dots, X_{t-1}) = P(X_t = q_j | X_{t-1})$$

- *Indépendance du temps* : la probabilité de passage d'un état à un autre ne varie pas avec le temps ; cela est formellement exprimé comme suit (*toujours pour un processus du premier ordre*):

$$P(X_t = q_j | X_{t-1} = q_i) = a_{ij} \text{ avec } a_{ij} \geq 0 \; et \; \sum_{j=1}^{N} a_{ij} = 1$$

Tel que : $1 \leq i, j \leq N$

Dans ce qui suit, on dénote par A la matrice des probabilités de transition.

Une chaine de Markov peut être alors vue comme un *automate probabiliste*. Différemment d'un automate ordinaire, l'état initial (*à l'instant t=1*) peut être n'importe quel élément dans S. Plus formellement, on définit la loi de probabilité de l'état initial $\Pi = \{\pi_i\}$ tel que [2,4]:

$$\pi_i = P(X_1 = q_i) \; avec \sum_{i=1}^{N} \pi_i = 1$$

L'*évolution* d'un modèle de Markov au cours du temps peut être schématisée par un treillis [2] comme dans la figure ci-dessous (*sur l'axe vertical se trouvent les états*).

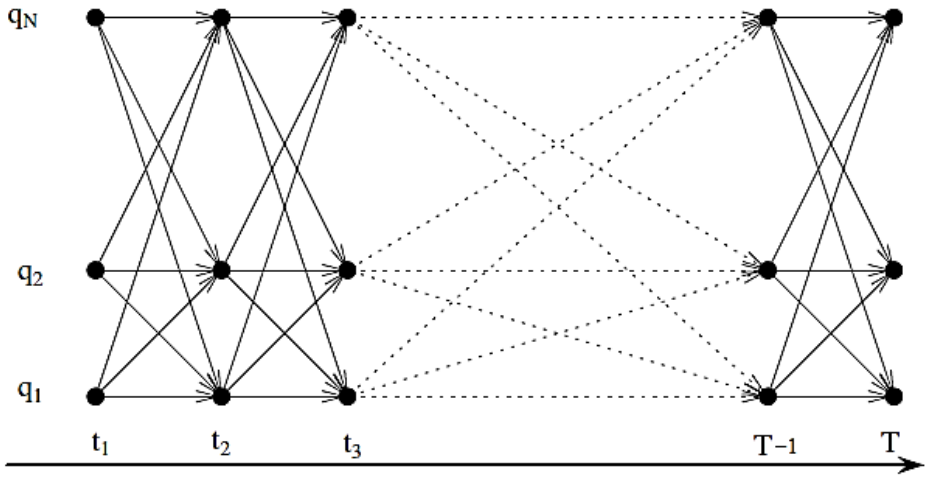


Figure 5.2 Représentation du modèle de Markov par un Treillis.

5.6.1 Modèles de Markov Cachés

Dans un modèle de Markov caché, la séquence d'états est inobservable (*d'où le nom caché du modèle*). Ce qu'on peut plutôt observer est la séquence des symboles émis par les états et qu'on dénote : $O = O_1, O_2, \dots, O_T$.

Un modèle de Markov caché [2,4] est défini par un quintuple $< S, V, \Pi, A, B >$ où $V = \{v_1, v_2, \dots, v_M\}$ est l'alphabet des symboles émis par les états. B dénote la

matrice des probabilités d'émission ; la probabilité que l'état q_j emet le symbole v_k, $b_j(v_k)$, est définie comme suit :

$$b_j(v_k) = P(o_t = v_k | X_t = q_j)$$

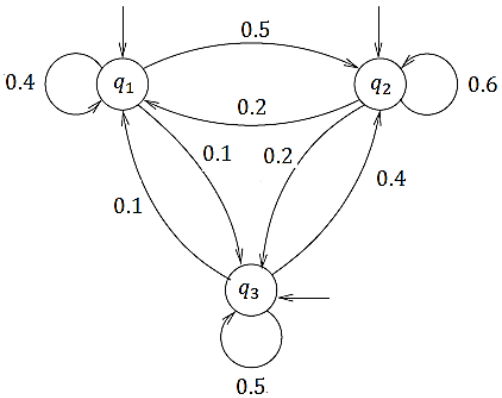
S, Π et T sont comme déjà définis dans la section précédente.

Exemple

Considérons l'expérience aléatoire qui consiste à lancer trois pièces de monnaie biaisées. Définissons les éléments du modèle de Markov caché correspondant:

- S= $\{q_1, q_2, q_3\}$ tel que : $q_1 : pièce\ 1, q_2 : pièce\ 2, q_3 : pièce\ 3$
- La matrice de transition $A = \begin{bmatrix} 0.4 & 0.5 & 0.1 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.4 & 0.5 \end{bmatrix}$

On peut schématiser les transitions entre les états par l'automate suivant :



- $V = \{P, F\}$ (P : Pile, F : Face)
- La matrice des probabilités d'émission $B = \begin{bmatrix} 0.48 & 0.52 \\ 0.80 & 0,20 \\ 0.55 & 0.45 \end{bmatrix}$
- Les probabilités de l'état initial : $\pi_1 = \pi_2 = 0.3\ \pi_3 = 0.4$

Imaginons que nous avons observé la séquence des symboles $O = O_1, O_2, \ldots, O_7$ =FPPFPPP, la question qui se pose est « quelle est la suite d'états ou bien le chemin la/le plus probable ayant généré cette séquence ?».

Afin de répondre à une telle question, on peut procéder d'une manière très naïve à calculer la probabilité de tous les chemins pouvant générer cette séquence, et puis choisir celui dont la probabilité est maximale. Cette solution est clairement très coûteuse, il existe heureusement des algorithmes plus appropriés. Particulièrement, nous nous intéressons dans ce qui suit à l'*algorithme de Viterbi*. Ce dernier est vu comme une application du principe de la *programmation dynamique*.

5.6.2 Algorithme de Viterbi

Etant donnés une observation et un modèle de Markov caché en entrée, l'algorithme de Viterbi permet de calculer la séquence d'états *optimale* qui a généré cette observation. Cette problématique est souvent connue par *la problématique de décodage*. Cet algorithme est souvent appliqué en TALN dans le cadre de *l'étiquetage morphosyntaxique* et dans celui de la *reconnaissance de la parole*. Définissons quelques notations [2]:

Etant donné un modèle de Markov caché, on défini $\lambda = (A, B, \Pi)$.
Notons par $\delta_t(j)$ la probabilité maximale d'une observation $o_1, o_2, \dots, o_t$ sous la condition que l'état courant soit q_j à l'instant t :

$$\delta_t(j) = \max_{s_1, s_2, \dots, s_{t-1}} P(s_1, s_2, \dots, s_{t-1}, o_1, o_2, \dots, o_t, s_t = q_j | \lambda)$$

$\psi_t(j)$ est le chemin optimal correspondant.

Algorithme de Viterbi (adapté de la référence [2])

1. Initialisation pour t=1
   Pour i=1 à N faire
   $\delta_1(i) = \pi_i \times b_i(o_1)$
   $\psi_1(i) = null$
   Fin Pour
2. Boucle d'induction
   t =2
   Tant que $t \leq T$ faire
   j= 1
   Tant que $j \leq N$ faire
   $\delta_t(j) = b_j(o_t) \times \max_{1 \leq i \leq N} \delta_{t-1}(i).a_{ij}$
   $\psi_t(j) = \arg \max_{1 \leq i \leq N} \delta_{t-1}(i).a_{ij}$
   $j = j + 1$
   $Fin\ TQ$
   t=t+1
   $Fin\ TQ$
3. Terminaison
   $\hat{P} = \max_{1 \leq i \leq N} \delta_T(i)$
   $\widehat{S_T} = \arg \max_{1 \leq i \leq N} \delta_T(i)$

Le chemin optimal est déterminé par un retour en arrière :
$\widehat{S_T}, \psi_{T-1}(\widehat{S_T}), \psi_{T-2}(\widehat{S_{T-1}}), \dots$

5.7 Traitement phonétique

Le traitement phonétique [2] concerne la production et la perception des sons. On appelle *phonème* l'unité acoustique minimale qui peut être une voyelle ou une consonne. La concaténation des phonèmes donne lieu aux *syllabes* qui construisent les mots. La capture des sons se fait à l'aide d'un microphone ce qui produit des signaux comme celui de la figure ci-dessous.

Figure 5.3 Un signal de la parole correspondant à "this is" [2].

En phonétique, on distingue deux types de traitements [2]:

- La *synthèse vocale* : ce traitement consiste à créer de la parole artificielle à partir du texte.
- La *reconnaissance de la parole* : il s'agit d'analyser la voix humaine afin de la transformer en texte exploitable par la machine.

Rapportons ici un exemple, pris de la référence [2], qui illustre ***le problème d'ambiguïté*** qui peut survenir dans la *reconnaissance de la parole*. Notons qu'on commence d'abord par la reconnaissance des phonèmes, des syllabes et ensuite des mots.

Soit la phrase «***The boys eat the sandwiches***». Partons des mêmes phonèmes, un système de reconnaissance vocale peut produire des résultats différents comme :

- ✓ The boy seat the sandwiches.
- ✓ The boy seat this and which is.
- ✓ The boys eat this and which is.
- ✓ The boys eat the sand which is…

5.8 Traitement morphologique

La morphologie [9] a pour objet l'étude des morphèmes et de leurs modes de combinaison. Un morphème est l'unité linguistique minimale ayant un sens (*un morphème n'est donc pas forcément un mot*). Voici des exemples de morphèmes:

	Mot	**Morphème**
Pomme de terre	Pomme De Terre	Pomme de terre
Livres	Livres	Livre s (*désignant le pluriel*)

On distingue deux types de morphèmes [2,9]:

- *Morphèmes lexicaux (dits aussi lexèmes):* il s'agit des quatre catégories noms, verbes, adjectifs et adverbes.
- *Morphèmes grammaticaux*: incluent les catégories des articles, des prépositions, des conjonctions, des verbes auxiliaires. A cela s'ajoutent les morphèmes qui caractérisent le genre et le nombre des noms, les temps de conjugaison et les personnes des verbes.

Les mots dans une phrase peuvent être annotés par leurs catégories lexicales, voici un exemple [2]:

Le	**gros**	**chat**	**mange**	**la**	**souris**	**grise**
article	adjectif	nom	verbe	article	nom	adjectif

On distingue habituellement deux modes de ***combinaison de morphèmes [9]***:

- *La composition* : qui consiste à concaténer des morphèmes lexicaux comme dans: chou-fleur, portemanteau, etc.
- *L'affixation :* il s'agit de la composition des mots en faisant interagir des morphèmes lexicaux (les racines) et des morphèmes grammaticaux (les affixes).

Pour les affixes, on peut discriminer [9]:

- *Les affixes dérivationnels* : comme les préfixes et les suffixes (*en résultat en crée de nouveaux mots*)
- *Les affixes flexionnels* : la flexion est la variation d'un mot sous certaines conditions grammaticales notamment le genre, le nombre, le temps, etc. Alors, les affixes flexionnels sont les morphèmes qu'on ajoute pour exprimer telle ou telle variation (*en résultat, on ne crée pas de nouveaux mots*).

Exemple

Mot	Racine	Préfixe dérivationnel	Suffixe dérivationnel	Suffixe flexionnel
intolérables	tolér	in	able	s

Les règles morphologiques permettent de générer toutes les formes de mots à partir d'un lexique. Les analyseurs morphologiques font l'opération inverse et récupèrent la racine du mot et ses différents affixes (dérivationnels ou flexionnels). Les parseurs morphologiques utilisent généralement les *automates à état fini* [2].

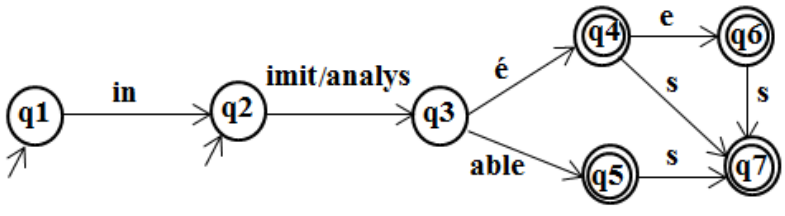
5.8.1 Automates à état fini

L'automate est un modèle de l'informatique théorique qui permet de représenter les *découpages morphologiques* sous la forme de règles**.** Cela permet de prédire la forme des mots qui peuvent apparaitre dans une langue [9].
Un automate à état fini A est défini par $A =< Q, V, \delta >$:

- Q est un ensemble fini d'états avec au moins un état initial et un état final.
- V est un vocabulaire fini. Dans notre contexte, il est constitué de l'ensemble des morphèmes considérés.
- δ est une fonction de transition définie depuis $Q \times V$ vers Q.

Les automates finis reconnaissent ou engendrent selon le point de vue adopté une classe particulière de langages appelée langages réguliers ou langages rationnels. Tout langage régulier peut être exprimé sous la forme d'une ou de plusieurs expressions régulières (et vice versa). Une expression régulière est une suite de symboles construite à partir d'un alphabet Σ et de la chaîne vide ε, à l'aide des opérations de concaténation (.) , de disjonction (|) et de la clôture de Kleen « * » . L'avantage des automates finis et des expressions régulières vient de ce qu'ils permettent de caractériser un nombre infini de mots différents à l'aide d'un nombre fini d'éléments [9].

Exemple



L'automate ci-avant reconnait le langage défini par l'expression régulière suivante :

$$(\text{in} \mid \epsilon).(\text{imit} \mid \text{analys}).((\text{é}.(\epsilon \mid \text{e} \mid \text{s} \mid \text{es})) \mid \text{able}(\epsilon \mid \text{s}))$$

L'implémentation de cet automate peut être faite via un programme Prolog comme suit:

morph.pl

File Edit Browse Compile Prolog Pce Help

morph.pl

```
init(q1).
init(q2).
fin(q4).
fin(q5).
fin(q6).
fin(q7).
transition(q1,in,q2).
transition(q2,imit,q3).
transition(q2,analys,q3).
transition(q3,é,q4).
transition(q4,e,q6).
transition(q3,able,q5).
transition(q4,s,q7).
transition(q5,s,q7).
accepte([],Etat):- fin(Etat).
accepte([Symbole|Symboles],Etat):-
        transition(Etat,Symbole,EtatSuivant),accepte(Symboles,EtatSuivant).
reconnaissance(Symboles):- init(InitEtat),accepte(Symboles,InitEtat).
```

Cet automate peut être utilisé en génération en utilisant la requête ? $-reconnaissance(L)$. (*tous les mots possibles sont énumérés*) ou en reconnaissance (*les mots appartenant au langage sont acceptés et les autres sont rejetés*) en utilisant une requête comme ? $-reconnaisance([in, analys, é])$.

```
?- reconnaissance(L).
L = [in, imit, é] ;
L = [in, imit, é, e] ;
L = [in, imit, é, s] ;
L = [in, imit, able] ;
L = [in, imit, able, s] ;
L = [in, analys, é] ;
L = [in, analys, é, e] ;
L = [in, analys, é, s] ;
L = [in, analys, able] ;
L = [in, analys, able, s] ;
L = [imit, é] ;
L = [imit, é, e] ;
L = [imit, é, s] ;
L = [imit, able] ;
L = [imit, able, s] ;
L = [analys, é] ;
L = [analys, é, e] ;
L = [analys, é, s] ;
L = [analys, able] ;
L = [analys, able, s] ;
```

```
?- reconnaissance([in,analys,é]).
true .

?- reconnaissance([in,analys]).
false.
```

5.8.2 Etiquetage morphosyntaxique avec l'algorithme de Viterbi

L'étiquetage morphosyntaxique consiste à faire associer à chaque mot sa catégorie grammaticale. Dans le tableau ci-dessous, se trouve une projection du modèle de Markov Caché sur cette application [2] :

MMC	Application en étiquetage morphosyntaxique
S	Ensemble des catégories morphosyntaxiques.
V	Ensemble des mots (le vocabulaire).
O	Une séquence de T mots observés qui prennent leurs valeurs dans V.
A	Probabilités que deux catégories morphosyntaxiques se suivent.
B	Probabilités d'observer un mot étant donnée une catégorie morphosyntaxique.
Π	La probabilité de débuter par une catégorie morphosyntaxique.

Exemple d'application (Reformulé à partir de la référence [4]).

Donnons tout d'abord, la liste des catégories morphosyntaxiques utilisées dans cet exemple :

Symbole	*Description*
$< s >$	Début d'une phrase
VB	Verb
NN	Noun
TO	infinitive marker
PPSS	nominative pronoun (comme: I, they,we)

Le MMC est défini par les éléments suivants :

- $S = \{< s >, VB, TO, NN, PPSS\}$
- $V = \{I, to, want, race\}$
- $P(X_1 = < s >) = 1$ (*un seul état initial possible*)
- La matrice A est définie comme suit :

	VB	**TO**	**NN**	**PPSS**
$< s >$	0.019	0.0043	0.041	0.067
VB	0.0038	0.035	0.047	0.007
TO	0.83	0	0.00047	0
NN	0.004	0.016	0.087	0.0045
PPSS	0.23	0.00079	0.0012	0.00014

- La matrice B est définie comme suit :

	I	***want***	***to***	***race***
VB	0	0.0093	0	0.00012
TO	0	0	0.99	0
NN	0	0.000054	0	0.00057
PPSS	0.37	0	0	0

NB. Les probabilités dans A et B sont calculées à base du corpus Brown.

Considérons la suite des mots observés : $0 = o_1, o_2, o_3, o_4, o_5 =< s >, \text{I}, \text{want}, \text{to}, \text{race}$.
Utiliser l'algorithme de Viterbi pour trouver la séquence d'états correspondante la plus probable.

Initialisation

$\delta_1(< s >) = 1\ \delta_1(PPSS) = 0\ \delta_1(VB) = 0\ \ \delta_1(TO) = 0\ \delta_1(NN) = 0$

$\psi_1(< s >) = Null\ \psi_1(PPSS) = Null\ \psi_1(VB) = Null\ \psi_1(TO) = Null\ \psi_1(NN) = Null$

Etape 2

$\delta_{PPSS}(I) = 0.37 \times 0.067 \times 1 \approx 0.025 \quad et \quad \psi_2(PPSS) =< S >$

$\delta_{VB}(I) = 0 \times 0.019 \times 1 = 0 \quad et \quad \psi_2(VB) =< S >$

$\delta_{TO}(I) = 0 \times 0.0034 \times 1 = 0 \quad et \quad \psi_2(TO) =< S >$

$\delta_{NN}(I) = 0 \times 0.41 \times 1 = 0 \quad et \quad \psi_2(NN) =< S >$

Etape 3

$\delta_{VB}(want) = 0.0093 \times \max(\mathbf{0.025} \times \mathbf{0.23}, 0 \times 0.0038, 0 \times 0.83, 0 \times 0.004) \approx 0.000053$
et $\psi_3(VB) = PPSS$
$\delta_{NN}(want) = 0.000054 \times \max(\mathbf{0.025} \times \mathbf{0.0012}, 0 \times 0.047, 0 \times 0.00047, 0 \times 0{,}0087)$
$= 0.162 \times 10^{-8}$ *et* $\psi_3(NN) = PPSS$
$\delta_{PPSS}(want) = 0$ et $\psi_3(PPSS) = PPSS$
$\delta_{TO}(want) = 0$ et $\psi_3(TO) = PPSS$
Etc.

5.9 Analyse syntaxique

Le niveau de la syntaxe explique comment mettre bout à bout des unités lexicales afin de bâtir des énoncées dont le sens global est plus que la simple somme des sens de ces unités [9]. Les travaux de *Noam Chomsky*, linguiste américain, sont sans doute les plus influençant au niveau de la syntaxe.

La théorie de *Chomsky* [2] suggère que « *la syntaxe est indépendante de la sémantique et peut être exprimée en terme de grammaires formelles* ». Ces grammaires consistent en un ensemble de règles qui décrivent la structure d'une phrase dans un langage. En outre, les règles peuvent générer la totalité des phrases (*dont le nombre peut être infini*) d'un langage donné. Les grammaires nous permettent alors de mener des opérations d'analyse syntaxique et de génération.

5.9.1 Grammaires formelles

Une grammaire formelle **G** est définie par le quadruplet $< \boldsymbol{V}, \boldsymbol{N}, \boldsymbol{P}, \boldsymbol{S} >$ tel que :

- V dénote le vocabulaire ou le lexique (ensemble des symboles terminaux).
- N dénote les symboles non-terminaux y compris le symbole initial S qui sert de point de départ.
- P est l'ensemble des règles de production ou de réécriture.

Les grammaires que nous utilisons dans ce cours sont des grammaires hors contexte (*dites aussi d'ordre 2*) dont les règles de production sont de la forme : $X \rightarrow \beta$ tel que X est un symbole non terminal et β est une séquence de symboles quelconques (terminaux ou non).

Prenons l'exemple d'une grammaire décrivant un sous ensemble très limité de l'anglais ou du français. Cette grammaire contient les règles suivantes [2]:

- ✓ Une phrase (S) consiste d'un groupe nominal (GN) et d'un groupe verbal (GV).
  $S \rightarrow GN\ GV$
- ✓ Un groupe nominal consiste d'un déterminant (Det) et d'un nom
  $GN \rightarrow Det\ Nom$
- ✓ Un groupe verbal consiste d'un verbe suivi d'un groupe nominal.
  $GV \rightarrow Verbe\ GN$

Si on considère le lexique ou le vocabulaire composé des déterminants (le, la), des noms (garçon, balle) et du verbe (frappe) alors on doit rajouter les règles :

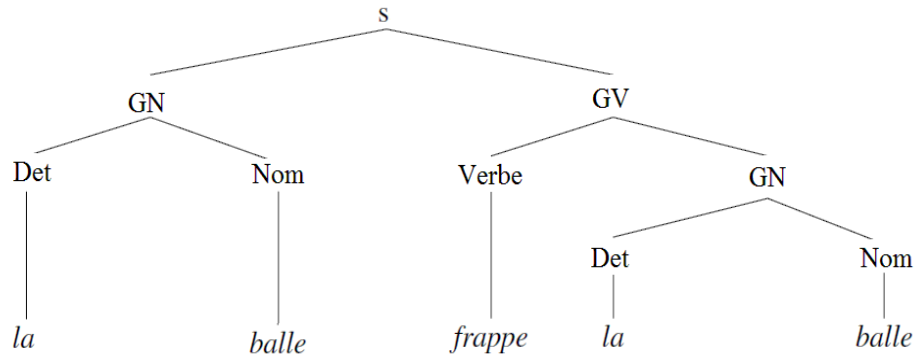
$Det \rightarrow le|la$
$Nom \rightarrow garçon \mid balle$
$Verbe \rightarrow frappe$

A partir des règles précédentes, on peut *générer* les phrases suivantes qui sont toutes syntaxiquement correctes (bien formées):

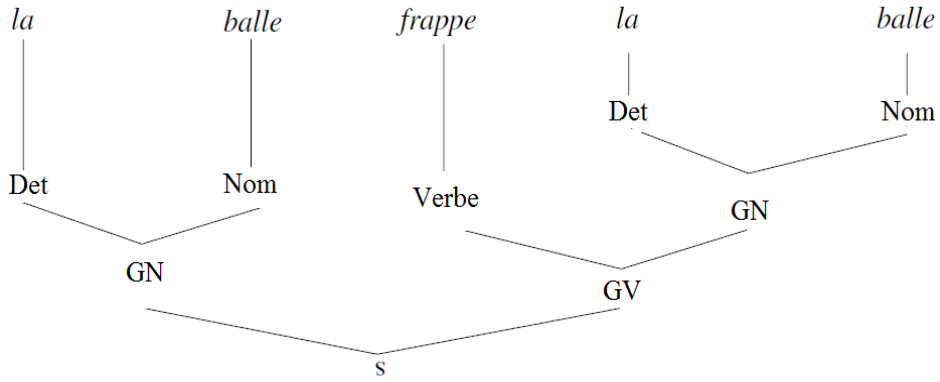
- ✓ La garçon frappe le balle
- ✓ La balle frappe le garçon
- ✓ Le garçon frappe la balle….

Cette grammaire nous permet aussi de construire un *arbre d'analyse* pour la phrase « la balle frappe la balle » :

- ➢ Par une approche *Top-down [3]* (en commençant par les règles)



- ➢ Par une approche *Bottom-up [3]* (en commençant par les mots de la phrase)



Le formalisme syntaxique de Chomsky est basé sur le concept des syntagmes. On appelle syntagme un groupe de mots ayant une signification et une même fonction syntaxique. Pratiquement, un syntagme correspond à un sous arbre dans l'arbre d'analyse syntaxique. Dans l'exemple précédent, « la balle » et « frappe la balle » sont des syntagmes mais pas « la balle frappe».

L'*ambiguïté syntaxique* se manifeste principalement dans les scénarios suivants [9]:

- Une même succession de catégories grammaticales qui ne donne pas un même arbre d'analyse. Par exemple :
  L'élève met les cahiers sur la table.
  Les enfants adorent les gâteaux avec du chocolat.
- Une même phrase ayant plusieurs interprétations peut être analysée différemment. Par exemple :
  L'homme observe l'enfant avec des jumelles.

L'analyse syntaxique n'est pas indépendante de l'analyse sémantique. En effet, pour analyser correctement une phrase (surtout en cas d'ambiguïté) il faut revenir absolument au sens désiré.

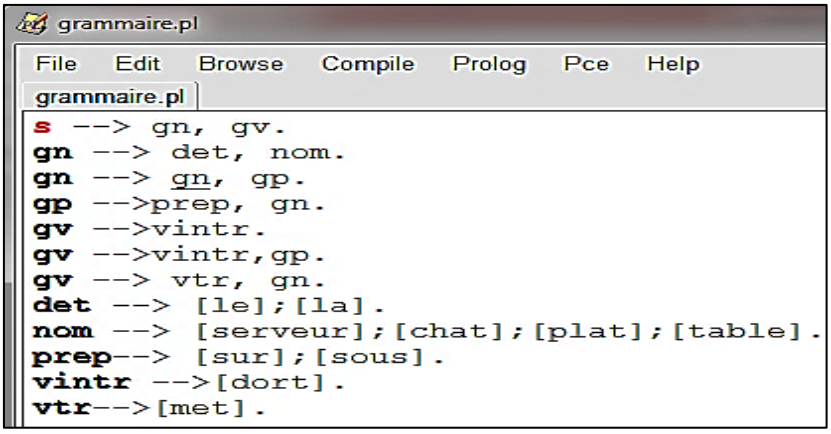
5.9.2 Grammaires à Clauses Définies en Prolog

La traduction des règles de grammaires à des règles DCG (*Definite Clause Grammars*) est directe. Cette notation utilise l'opérateur $--> $ pour exprimer qu'un syntagme peut être composé d'une séquence de syntagmes plus simples [2]. Une fois le programme Prolog écrit en notation DCG est chargé en mémoire, il est systématiquement traduit en clauses Prolog [3]. A savoir, une clause est une affirmation inconditionnelle (*i.e. fait*) ou conditionnelle (*i.e. règle*).

Illustrons dans ce qui suit l'utilisation du Prolog en analyse syntaxique à travers un exemple. Dans cet exemple, nous introduisons de nouveaux symboles préposition (Prep), groupe prépositionnel (GP), verbe intransitif (Vintr) et verbe transitif (Vtr). Soit la grammaire $G = < V, N, P, S >$, tel que :

$V = \{le, la, serveur, chat, plat, table, dort, met, sur, sous\}$
$N = \{S, GN, GV, GP, Det, Nom, Prep, Vtr, Vintr\}$
P={ $S \rightarrow GNGV$, $GN \rightarrow Det\ Nom$, $GN \rightarrow GN\ GP$, $GP \rightarrow Prep\ GN$, $GV \rightarrow Vintr$, $GV \rightarrow Vintrn\ GP$, $GV \rightarrow Vtr\ GN$, $Det \rightarrow le|la$, $Nom \rightarrow serveur|chat|plat|table$, $Prep \rightarrow sur|sous$, $Vintr \rightarrow dort$, $Vtr \rightarrow met$}

Le programme prolog, écrit en notation DCG, correspondant à cette déclaration est montré dans la figure ci-dessous :



Remarque

Au fait, chaque non terminal définit un prédicat d'arité 2. Par exemple, la règle s $--> $ gn, gv est équivalente à la clause suivante [2]:

$$s(L1, L) :- gn(L1, L2), gv(L2, L).$$

Cette dernière est en réalité équivalente à [2]:

$$s(L) :- gn(L1), gv(L2), append(L1, L2, L).$$

L'interpréteur Prolog nous permet de chercher toutes les phrases bien formées générées par cette grammaire en utilisant la syntaxe suivante :

?- s(L,[]).
L= [le, serveur, dort] ;
L= [le, chat, dort, sur, le, plat] ;
L= …

On peut tester la grammaticalité des phrases, comme dans les requêtes suivantes :

```
?- s([le,chat,dort,sous,la,table],[]).
true .

?- s([le,chat,dort],[]).
true .

?- s([le,serveur,met,le,plat,sur,la,table],[]).
true .
```

On peut consulter la trace du processus d'analyse mené par Prolog (*qui utilise une approche top-down*). Voici un exemple :

```
[trace]  ?- s([le,chat,dort],[]).
   Call: (8) s([le, chat, dort], []) ? creep
   Call: (9) gn([le, chat, dort], _2608) ? creep
   Call: (10) det([le, chat, dort], _2608) ? creep
   Call: (11) [le, chat, dort]=[le|_2594] ? creep
   Exit: (11) [le, chat, dort]=[le, chat, dort] ? creep
   Exit: (10) det([le, chat, dort], [chat, dort]) ? creep
   Call: (10) nom([chat, dort], _2614) ? creep
   Call: (11) [chat, dort]=[serveur|_2600] ? creep
   Fail: (11) [chat, dort]=[serveur|_2600] ? creep
   Redo: (10) nom([chat, dort], _2614) ? creep
   Call: (11) [chat, dort]=[chat|_2600] ? creep
   Exit: (11) [chat, dort]=[chat, dort] ? creep
   Exit: (10) nom([chat, dort], [dort]) ? creep
   Exit: (9) gn([le, chat, dort], [dort]) ? creep
   Call: (9) gv([dort], []) ? creep
   Call: (10) vintr([dort], []) ? creep
   Exit: (10) vintr([dort], []) ? creep
   Exit: (9) gv([dort], []) ? creep
   Exit: (8) s([le, chat, dort], []) ? creep
true
```

A cause de la règle récursive à gauche : gn $--->$ gn, gp, une requête portant sur une phrase syntaxiquement incorrecte entrainera une boucle infinie et donc un message d'erreur sera affiché en réponse. Pour se débarrasser de la récursivité, on peut ajouter un symbole auxiliaire [2] comme suit :

ngroupe $--->$ det, nom.
gn $--->$ ngroupe.
gn $--->$ ngroupe, gp.

Dans ce cas, Prolog échoue (renvoie false), comme le confirment les requêtes suivantes :

```
?- s([le,chat,le,plat],[]).
false.

?- s([le,plat,le,serveur,la,table],[]).
false.

?- s([le,chat,le,serveur,la,table,le,plat],[]).
false.
```

5.10 Sémantique Lexicale

La sémantique lexicale est l'étude du sens des morphèmes d'une langue. Cette définition est en réalité assez problématique puisque pour définir le "sens" d'un mot on fait recours à d'autres mots (*circularité du sens*) [9]. La sémantique lexicale est très importante pour les applications du TALN, comme en [10]:

- *Traduction automatique* : un même mot peut avoir différentes traductions. Par exemple, le verbe « fuit » en français, selon le contexte doit être traduit en arabe à « يهرب » ou «يتسرب»
- *Recherche des documents* : on peut éviter le silence en exploitant les synonymes et éviter le bruit en se référant au contexte.
- *Génération de texte* pour choisir les bonnes collocations (*une collocation est une association habituelle d'un mot à un autre au sein d'une phrase*). Par exemple : il faut dire « réaliser un souhait » au lieu de « effectuer un souhait », « tenir une promesse » au lieu de « satisfaire une promesse », etc.

5.10.1 Décomposition en primitives sémantiques

L'analyse sémique [9] est une des premières tentatives de décomposition du sens d'un mot en unités de sens élémentaires : appelées *sèmes*. Un sème est un *trait sémantique minimal* dont les seules valeurs possibles sont : positif (+), négatif (-) ou sans objet (Ø). Un ***sémème*** est un faisceau de sèmes correspondant à une unité lexicale.

Naturellement, les mots ayant des sèmes positifs communs appartiennent au même champ sémantique. Pour illustrer cette idée, on s'inspire ici d'un exemple classique d'analyse sémique dû à Bernard Pottier et qui porte sur les sièges. D'après le dictionnaire Larousse un siège est un: « *Meuble ou tout autre objet fait pour s'asseoir* ». En ce sens, fauteuil, tabouret et chaise sont des sièges comme le montre les triangles sémiotiques ci-dessous :

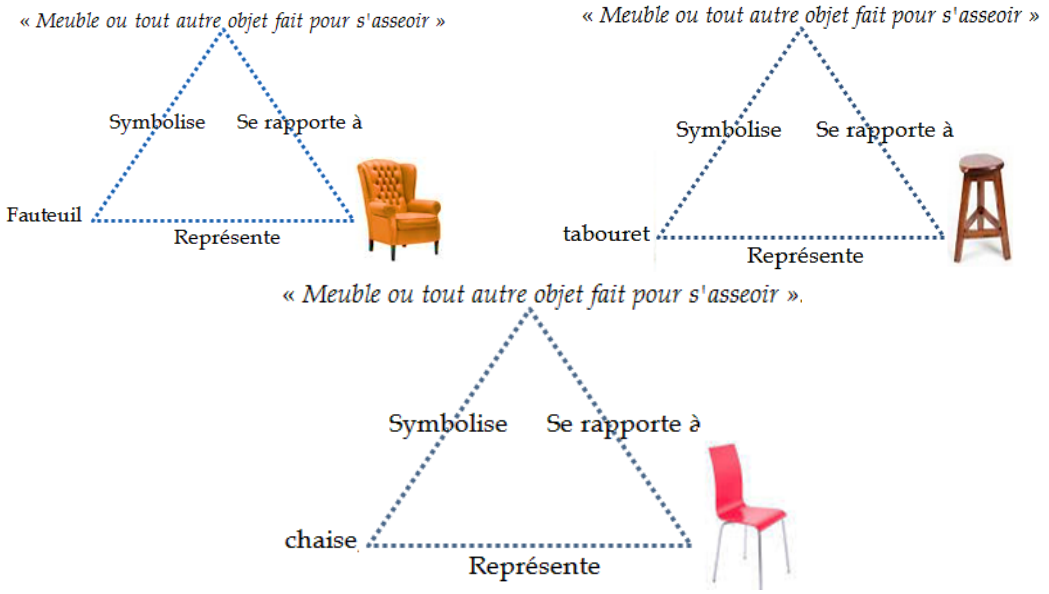


Figure 5.4 Triangles sémiotiques pour fauteuil, tabouret et chaise.

On veut effectuer une analyse sémique pour ces mots. Dans cette fin, on se donne les traits sémantiques :

- ✓ S1 : pour s'asseoir.
- ✓ S2 : pour une personne.
- ✓ S3 : avec dossier.
- ✓ S4 : avec bras.

Cela donne la matrice d'analyse suivante :

	S1	S2	S3	S4
Fauteuil	+	+	+	+
Tabouret	+	+	-	-
Chaise	+	+	+	-

5.10.2 Polysémie et ambiguïté

Par polysémie, on désigne la pluralité du sens liée à un même sémème. C'est la mise en contexte (ou encore en discours) qui permettra de lever l'ambiguïté qui en découle. Pour expliquer cela, nous nous servons de l'analyse sémique du mot canard proposé par Katz et Fodor [9] qu'ils ont présenté sous forme d'arborescence.

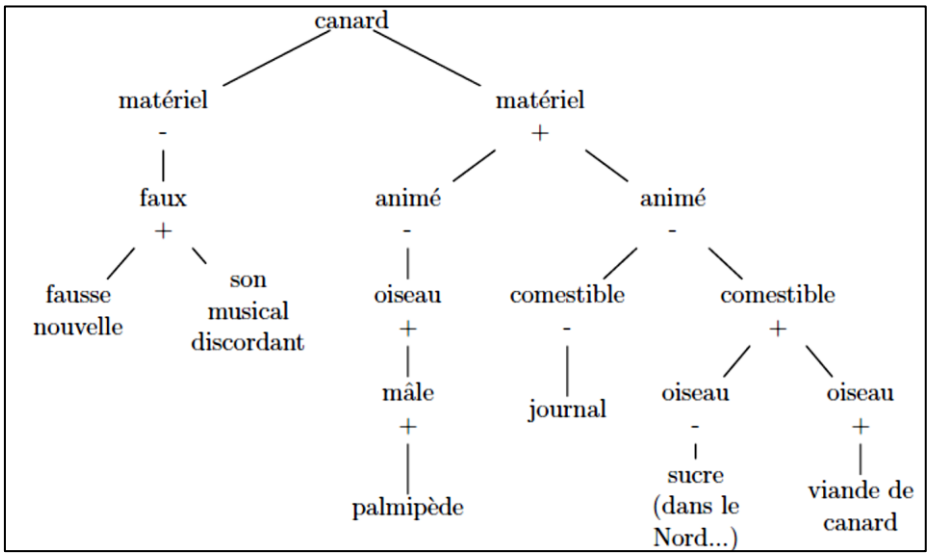


Figure 5.5 Analyse sémique du mot **canard** (schéma de Katz et Fodor) [9].

Montrons comment cette analyse sémique peut être exploitée dans la désambiguïsation du mot canard dans la phrase suivante [9]:

« Il tenait le canard dans ses mains.... malheureusement le canard s'envola».

Le verbe tenir contraint son complément d'objet à avoir un trait matériel. Cela permet d'éliminer la branche (non matériel). S'envola vérifie le trait animé alors l'ambiguïté est levée : il s'agit d'un oiseau. Mais est-ce que cette analyse est suffisante pour lever l'ambiguïté apparaissant dans une phrase comme : *« Il continue à lancer des canards »* ?

5.10.3 Similarité entre mots d'après thésaurus

Parmi les *relations sémantiques* pouvant exister entre les mots, la *synonymie* est la relation la plus étudiée puisque elle a plusieurs applications pratiques. La *relation*

de similarité est une version assouplie de la synonymie où on parle plutôt d'un degré de similarité entre deux mots [4].

Le calcul de similarité des mots est utile pour de nombreuses applications du TALN comme [4] : la recherche d'information, les systèmes questions-réponses, le résumé automatique, la traduction automatique, etc. Dans les métriques basées sur thésaurus, la distance entre deux mots est déterminée à partir d'un thesaurus électronique comme WordNet généralement en exploitant la relation de subsomption (est-un). Voici des exemples de ces métriques [4]:

- *Similarité basée sur la longueur de chemin* : Plus le chemin entre deux mots dans le graphe défini par la hiérarchie du thésaurus est court, plus ils sont similaires. Cette notion peut être opérationnalisée en mesurant le nombre d'arêtes entre les deux mots dans le graphe thésaurus. Naturellement, la similarité est inversement proportionnelle à la longueur du chemin. La définition la plus utilisée pour cette métrique est la suivante ($m_1 et\ m_2$ sont des mots, *L dénote la longueur de chemin*):

$$sim_{chemin}(m_1, m_2) = -\log L(m_1, m_2)$$

- *Similarité de Resnik :* l'intuition derrière cette métrique est que plus deux mots ont de contenu sémantique en commun, plus ils sont similaires. Le contenu sémantique d'un mot est défini comme le logarithme négatif de sa probabilité d'apparition dans un corpus. Cette probabilité est estimée à base des occurrences des mots subsumés par ce mot comme suit *(N est le nombre total des mots dans le corpus apparaissant également dans le thésaurus, mots(M) est l'ensemble des mots subsumés par M) :*

$$P(M) = \begin{cases} 1 & si\ M = Racine \\ \dfrac{\sum_{m \in mots(M)} fréquence(m)}{N} & Sinon \end{cases}$$

Etant donnés deux concepts, leur facteur commun en quelque sorte coïncide avec leur subsumant commun le plus spécifique (*LCS : Lowest Common Subsumer*). D'après Resnik, la similarité entre deux mots est égale au contenu sémantique de leur LCS comme le reflète la définition suivante:

$$sim_{Resnik}(m_1, m_2) = -\log P(LCS(m_1, m_2))$$

D'autres métriques qui sont conceptuellement proches à celle de Resnik sont les suivantes [4]:

- *Similarité de Lin :*

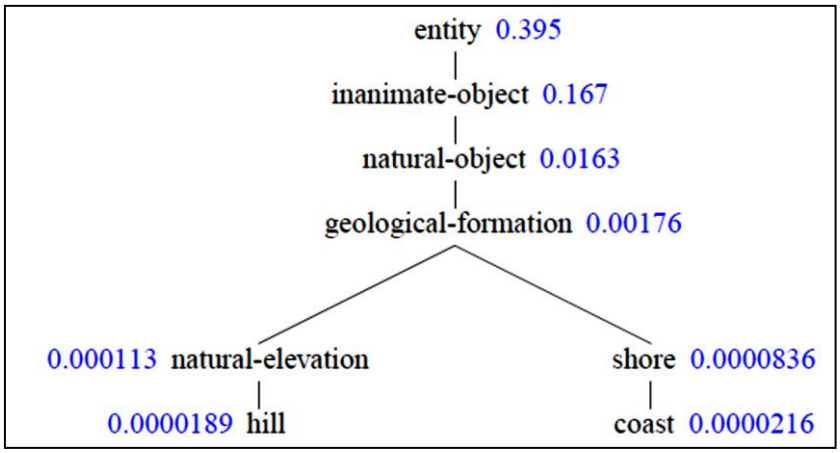
$$sim_{Lin}(m_1, m_2) = \frac{2 \log P(LCS(m_1, m_2))}{\log P(m_1) + \log P(m_2)}$$

- *Similarité de Jiang-Conrath :*

$$sim_{Jc}(m_1, m_2) = \frac{1}{2 \log P(LCS(m_1, m_2)) - (\log P(m_1) + \log P(m_2))}$$

Exemple

Soit le fragment d'hiérarchie WordNet ci-dessous où à chaque mot M est attachée sa probabilité d'apparition P(M) [4]:



Déterminer $L(hill, natural\_evaluation), L(hill, entity)$, $L(hill, shore), L(hill, coast)$.
Calculer la similarité entre les mots « $hill$ » et « $coast$ » , en utilisant les métriques déjà vues.

Réponse

$L(hill, natural\_evaluation) = 1$
$L(hill, entity) = 5$
$L(hill, shore) = 3$
$L(hill, coast) = 4$
$sim_{chemin}(hill, coast) = -\log L(hill, coast) = -\log 4 \approx -0.6$
$sim_{Resnik}(hill, coast) = -\log P(LCS(hill, coast))$

$$= -logP(geological - formation) \quad = -\log(0.00176) \approx 2.75$$

$$sim_{Lin}(hill, coast) = \frac{2 \times \log P(geological - formation)}{\log P(hill) + \log P(coast)}$$

$$= \frac{2 \times \log(0.00176)}{\log(0.0000189) + \log(0.0000216)} \approx 0.59$$

$$sim_{Jc}(hill, coast) = \frac{1}{2 \times \log P(geological - formation) - (\log P(hill) + \log P(coast))}$$

$$= \frac{1}{2 \times \log(0.00176) - (\log(0.0000189) + \log(0.0000216))} \approx 0.26$$

5.11 Conclusion

Le TALN est un des champs de l'IA les plus importants avec beaucoup d'applications pratiques. De nos jours, nous utilisons fréquemment des moteurs de recherche d'information, des traducteurs automatiques, des interfaces vocales, etc. Même si nous nous ne sommes pas complètement satisfaits de la qualité de ces applications, elles nous sont quand même d'une grande utilité.

A la fin de ce chapitre, l'étudiant doit réaliser la complexité du TALN mais il doit comprendre également les sources d'imperfections enregistrées dans ses applications même les plus contemporaines.

Travail Pratique

NLTK[3] (*Natural Language Toolkit*) est une plate-forme qui permet la création de programmes Python de TALN. Elle fournit des interfaces à plusieurs corpus et thésaurus ainsi qu'une suite de bibliothèques de traitement de texte pour la tokenization, l'étiquetage, l'analyse, le raisonnement sémantique, etc.

Installation

1. Télécharger /Installer Python (https://www.python.org/downloads/)
2. Télécharger /Installer NLTK (https://pypi.python.org/pypi/nltk)

Exploration du Corpus Brown

Importer le corpus Brown et faire lister les différentes catégories de textes qu'il contient en utilisant la syntaxe suivante [11]:

```
>>> from nltk.corpus import brown
>>> brown.categories()
```

Maintenant, on veut par exemple collecter quelques statistiques sur la catégorie «news». Nous nous intéressons au calcul de la fréquence des verbes modaux. Nous utilisons particulièrement la fonction ***FreqDist (...)*** [11]:

```
>>> from nltk.corpus import brown
>>> news_text = brown.words(categories='news')
>>> fdist = nltk.FreqDist([w.lower() for w in news_text])
>>> modals = ['can', 'could', 'may', 'might', 'must', 'will']
>>> for m in modals:
... print m + ':', fdist[m],
```

Dans le cours, nous avons mentionné que Brown est un corpus annoté. Nous voulons savoir quelle est la fréquence de chaque étiquette morphosyntaxique appartenant à *L'Universal Part-of-Speech Tagset* , toujours dans la catégorie *news* [11] *:*

Universal Part-of-Speech Tagset

Tag	Meaning
ADJ	adjective
ADP	adposition
ADV	adverb
CONJ	conjunction
DET	determiner, article
NOUN	noun
NUM	numeral
PRT	particle
PRON	pronoun
VERB	verb
.	punctuation marks
X	other

[3] http://www.nltk.org/

```
>>> from nltk.corpus import brown
>>> brown_news_tagged = brown.tagged_words (categories='news',
tagset='universal')
>>> tag_fd = nltk.FreqDist(tag for (word, tag) in brown_news_tagged)
>>> tag_fd.most_common()
[('NOUN', 30640), ('VERB', 14399), ('ADP', 12355), ('.', 11928), ('DET', 11389),
 ('ADJ', 6706), ('ADV', 3349), ('CONJ', 2717), ('PRON', 2535), ('PRT', 2264),
 ('NUM', 2166), ('X', 106)]
```

Exploration du Thésaurus Wordnet

Importer Wordnet et chercher les synonymes des mots « entity » et « coast » :

```
>>>from nltk.corpus import wordnet
>>>wordnet.synsets('entity')
…?
>>> wordnet.synsets(' coast')
…?
```

Mesures de Similarité[4]

Nous allons refaire l'exemple déjà vu dans le cours concernant le calcul des métriques de similarité entre mots :

- Déterminer la mesure de similarité basée sur les chemins entre les paires $(hill, natural\_evaluation)$, $(hill, entity)$, $(hill, shore)$, $L(hill, coast)$.
  Pour ce faire exploiter: **wordnet.path_similarity(…)**
- Calculer la similarité entre « $hill$ » et « $coast$ » en utilisant les métriques de :
  Resnik en utilisant **wordnet.res_similarity(…)**
  Lin en utilisant **wordnet.lin_similarity(…)**
  Jiang-Conrath en utilisant **jcn_similarity(…)**

Afin de pourvoir appliquer ces métriques, il faut d'abord collecter un **c**ontenu **i**nformationnelle (**ic**), dans notre exemple, à base du corpus Brown. Pour ce faire, on procède comme suit :

```
>>> from nltk.corpus import wordnet_ic
>>> brown_ic = wordnet_ic.ic('ic-brown.dat')
```

[4] http://www.nltk.org/howto/wordnet.html

Références du chapitre

[1] F. Yvon, *Une petite introduction au Traitement Automatique des Langues Naturelles* (https://perso.limsi.fr/anne/coursM2R/intro.pdf)

[2] P. Nugues, *Introduction to Language Processing with Perl and Prolog: An Outline of Theories, Implementation, and Application with Special Consideration of English, French, and German*, Springer, 2010.

[3] M. Covington, *Natural language processing for Prolog programmers*. Upper Saddle River N.J.: Prentice Hall, 2002.

[4] D. Jurafsky and J. Martin, *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. [India]: Dorling Kindersley Pvt, Ltd., 2014.

[5] B. Bigi, *Modélisation Statistique du Langage* (http://lpl-aix.fr/~bigi/Doc/bigi2006_02_ML1.pdf)

[6] B. Bigi, *Corpus* (http://lpl-aix.fr/~bigi/Doc/bigi2006_02_CORPUS.pdf)

[7] M. Scott, *Programming Language Pragmatics* (SECOND EDITION).Elsevier, 2006.

[8] S. Russell and P. Norvig, *Intelligence artificielle*. Paris: Pearson Education, 2010.

[9] I. Tellier , *Introduction au TALN et à l'ingénierie linguistique* (http://www.lattice.cnrs.fr/sites/itellier/poly_info_ling/info-ling.pdf)

[10] F. Gayral, *Sémantique Lexicale pour le TAL*, cours de l'option "Traitement automatique des langues" en master 2eme année MICR, Institut Galilée - Université Paris 13,2006.

[11] S. Bird, E. Klein and E. Loper, *Natural language processing with Python*. USA: O'Reilly Media, 2009.

Annexe : Sujets des Examens

Examen des Systèmes Intelligents
Département d'Informatique, Master II IA
Année universitaire 2015/2016

Exercice 1 [3 points]

1. Le niveau cognitif de NEWELL est un niveau de description des connaissances indépendant de toute représentation informatique. Une ontologie est codée en OWL alors sa spécification ne se fait pas au niveau cognitif. Vrai ou faux ? Argumenter !
2. Un agent implémentant un SLF de type TSK peut être vu comme un agent réactif. L'architecture de type réactive vue dans le cours est celle de subsomption. Alors, cet agent possède une architecture de subsomption. Vrai ou faux ? Argumenter !
3. Les morphèmes « Ontologie, Taxonomie, Vocabulaire, Thésaurus » appartiennent au même champ sémantique. Mettre en évidence cet énoncé par une analyse sémique: proposer au moins 4 sèmes différents tout en donnant une définition claire pour chacun.

Exercice 2 [4 points]

Imaginons qu'on dispose de deux ontologies décrivant : « les Enseignants, les Chercheurs ainsi que les Documents rédigés par ces derniers» :

- La première ontologie est construite selon le point de vue « tâche pédagogique» (utilisée par exemple au sein du département) ; Voici son ABOX complète [1]:

ABOX de la première ontologie		
Enseignant (Omar)	Enseignant (Ali)	Enseignant (Sonia)
Enseignant (Lila)	Enseignant (Dina)	
Enseignant Vacataire ()	Chargé de cours (Ali)	Chargé de TD (Lila)
Chargé de TP (Omar)	Chargé de TP (Sonia)	Chargé de TP (Dina)
Support de cours (doc3)	Support de cours (doc4)	

- La deuxième ontologie est construite selon le point de vue « grade scientifique» (utilisée par exemple au sein du laboratoire de recherche) ; Voici son ABOX complète [2] :

ABOX de la deuxième ontologie	
Maitre-assistant classe A (Omar)	Maitre-assistant classe A (Sonia)
Maitre de conférences (Lila)	Professeur (Ali)
Enseignant-chercheur (Omar)	Enseignant-chercheur (Ali)
Enseignant-chercheur (Sonia)	Enseignant-chercheur (Lila)
Article de recherche (doc1)	Article de recherche (doc2)

On veut fusionner les deux ontologies. Exprimer en LD la TBOX de l'ontologie fusionnée (résultante) tout en donnant les justifications formelles adéquates.

[1] [2] Tous les individus sont mentionnés.

Exercice 3 [5,5 points]

Considérons un contrôleur flou d'un ventilateur de maison qui fonctionne selon les règles suivantes :

Si Température est Faible **ou** Humidité est Sec	**Alors** Vitesse est Lente
Si Température est Moyenne **et** Humidité est Humide	**Alors** Vitesse est Moyenne
Si Température est Elevée	**Alors** Vitesse est Rapide

1. Déduire la spécification PEAS utilisée dans la conception de l'agent ventilateur flou.
2. Faire le parallèle entre le modèle de l'agent ventilateur flou et le modèle formel abstrait des agents : pour répondre à cette question déterminer à quoi correspondent les ensembles S, A, P et les fonctions See et Action.
3. Refaire la question précédente pour le cas d'un SLF de type TSK quelconque.

Exercice 4 [6,5 points]

Imaginons un robot aspirateur qui fonctionne par des ordres vocaux. Ce robot est capable d'éviter les obstacles et les vides. En outre, il est conscient de son niveau d'énergie. Considérons que :

- Initialement, le robot est rangé dans sa base.
- Le robot reconnait et obéit les sons « Begin » et « End ».
- Le robot navigue dans une pièce tout en nettoyant jusqu'à ce que la saleté soit définitivement éliminée.
- Le robot évite les obstacles en changeant de direction et évite les vides en reculant.
- Le robot commence à se recharger une fois un niveau critique de sa batterie est atteint.
- Le robot se range une fois sa tâche est terminée ou s'il reconnait « End ».
- Le robot se recharge et se range dans la même base.

1. Comment appelle-t-on Begin et End dans le vocabulaire du TALN ?
2. La sémantique de Begin et End est-elle vraiment acquise par le robot ? Expliquer !
3. Le choix d'une interface vocale pour la commande de l'agent est-il sans inconvénient ?
4. En s'inspirant de l'architecture de subsomption, établir les règles susceptibles de bien piloter notre robot aspirateur.
5. Exprimer en langue naturelle : une propriété de sûreté et une autre de vivacité (*de votre choix*) que le robot doit vérifier.

Question Indépendante [1 point]

Donner une structure de Kripke qui satisfait la formule CTL : EF (p v q) (*p et q sont des formules logiques atomiques*).

Examen des Systèmes Intelligents
Département d'Informatique, Master II IA
Année universitaire 2016/2017

Exercice 1 [Logique Temporelle – 2 Pts]

Soit une maison à trois étages plus un rez-de-chaussée (le rez-de-chaussée correspond à l'étage 0). A chaque étage, il y a une porte pour l'ascenseur. On définit les propositions suivantes :

O_i : La porte de l'étage i est ouverte

E_i : L'ascenseur est à l'étage i

Formaliser en LTL les deux propriétés suivantes:

- L'ascenseur toujours fini par retourner à l'étage 0.
- Une porte ne s'ouvre que si l'ascenseur est derrière la porte.

Exercice 2 [TALN, Ontologie, Logique de Description – 9 Pts]

Soit le texte suivant: « Le mot domotique vient de *domus* qui signifie domicile et de *tique* qui fait référence à la technique. La domotique est l'ensemble des techniques permettant de centraliser le contrôle des différents sous-systèmes de la maison (chauffage, porte de garage, portail d'entrée, etc.). La domotique vise à répondre aux besoins de confort (gestion d'énergie, optimisation de l'éclairage et du chauffage), de sécurité et de communication (commandes à distance, signaux visuels ou sonores, etc.)»

1. Déterminer les *morphèmes* constituant les mots: domotique, sous-systèmes.
2. Sachant qu'un groupe nominal peut être formé d'un nom sans déterminant, donner une *grammaire* permettant de dériver la phrase:

   « ***tique fait référence à la technique*** »
3. Donner *l'arbre syntaxique* correspondant à la phrase précédente.
4. Donner *la hiérarchie de concepts* qu'on peut construire à partir de ce texte.
5. Formaliser en *LD* la partie de la hiérarchie extraite de la dernière phrase.

Exercice 3 [Agents Intelligents, Système TSK, Système de MAMDANI – 9 Pts]

Ahmed possède une maison intelligente. Imaginons **l'agent « box-domotique »** qui, en fonction des informations obtenues via internet ou recueillies par les différents capteurs disséminés à travers la maison, commande: la porte du garage, le robot aspirateur et les stores.

I) L'agent contrôleur de l'ouverture de la porte du garage fonctionne selon les règles suivantes:

Si Ahmed travaille et Ahmed est chez lui Alors ouvrir la porte du garage à 7H30.
Si Ahmed ne travaille pas et Ahmed est chez lui Alors ouvrir la porte du garage à 10H30.
Si Ahmed ne travaille pas et Ahmed est chez lui et c'est Vendredi Alors ouvrir la porte du garage à 12H30.

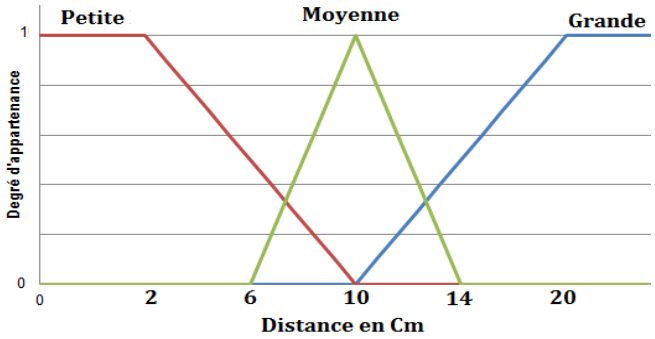
1. Quel est le *type de cet agent* : réactif ou proactif ? Justifier votre réponse.
2. Donner la *spécification PEAS* correspondante à cet agent.
3. Donner la *modélisation* correspondante à cet agent (*définir S, A, Action*).

II) Le **robot aspirateur** de Ahmed évite les obstacles en utilisant un système de type TSK d'ordre 0 qui en fonction de la distance de l'obstacle ajuste l'angle de rotation comme suit :

Si la distance est petite alors l'angle est -10°
Si la distance est moyenne alors l'angle est 0°
Si la distance est grande alors l'angle est +10°

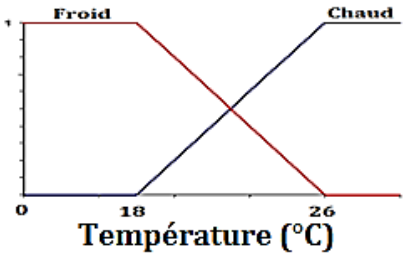
Question. Calculer l'angle de rotation si l'obstacle se trouve à une distance de :

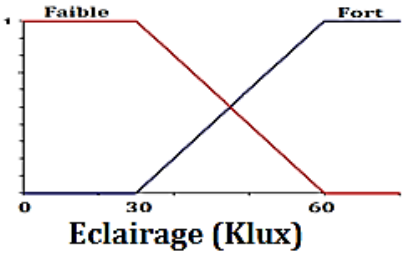
- 6.5 cm
- 9.5 cm
- 11.5 cm

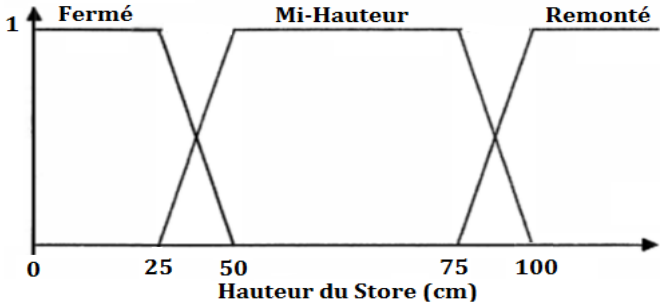


III) Dans la maison de Ahmed, la monté et la descente des stores est automatisée. Tous les stores sont gérés par un système de type MAMDANI qui a deux entrées : la température (°C) et l'éclairage (KLux) et comme sortie la hauteur du store (cm).

Si Température est Froid et Eclairage est Faible Alors Store est Remonté
Si Température est Chaud et Eclairage est Faible Alors Store est Remonté
Si Température est Froid et Eclairage est Fort Alors Store est Mi-Hauteur
Si Température est Chaud et Eclairage est Fort Alors Store est Fermé







Question. Calculer la hauteur du store si :

- Température = 3 et Eclairage = 50
- Température = 22 et Eclairage = 66
- Température = 25 et Eclairage = 45

Une dernière Question. A la lumière des données précédentes, déduire la spécification PEAS de l'agent **« box-domotique».**

Examen des Systèmes Intelligents
Département d'Informatique, Master II IA
Année universitaire 2017/2018

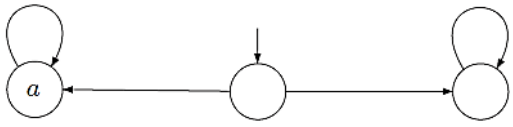
Exercice 1 (4 points)
Cocher **la** bonne réponse :

1. Le Système expert MyCin ne peut pas être considéré comme un agent puisque :
   - ☐ Il n'est pas rationnel
   - ☐ Il n'interagit pas directement avec un environnement
   - ☐ Il n'est pas intentionnel
2. « Désambiguïsation » est un :
   - ☐ Phonème
   - ☐ Morphème
   - ☐ Mot
3. Une règle de grammaire permet d'exprimer que:
   - ☐ Une phrase est composée de plusieurs morphèmes.
   - ☐ Un syntagme est composé de plusieurs syntagmes plus simples.
   - ☐ Aucune réponse n'est juste.
4. L'analyseur syntaxique écrit en Prolog vu dans le cours effectue une:
   - ☐ Analyse Top-Down
   - ☐ Analyse Bottom-up
5. La probabilité d'un mot calculée à partir d'un corpus est élevée quand:
   - ☐ Il a moins de contenu sémantique
   - ☐ Le nombre des occurrences de ses subsumant est petit
   - ☐ Aucune réponse n'est juste.
6. La logique LTL permet d'exprimer :
   - ☐ La précédence entre les évènements
   - ☐ Le temps précis des évènements
   - ☐ Les deux réponses sont correctes
7. La formule p doit être au moins une fois suivie par sa négation :
   - ☐ $F(p \wedge X\neg p)$
   - ☐ $pU\neg p$
   - ☐ $pUX\neg p$
8. La formule p doit arriver une seule fois :
   - ☐ $\neg p\ U\ (p \wedge XG\neg p)$
   - ☐ $F\ p$
   - ☐ $\neg G\neg p$

Exercice 2 (7 points)

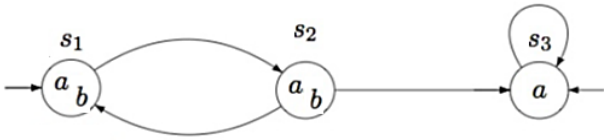
1. Montrer par déduction naturelle que : $\vdash_{KT45} \Box \Diamond \Box p \rightarrow \Box p$
2. Exprimer en LTL et en CTL que: « *Chaque requête finit par avoir une réponse* ».
3. Cocher la/les bonne(s) réponse(s) :

   Soit la structure de Kripke suivante $\mathcal{M}$:



   - ☐ $\mathcal{M} \vdash AFa$
   - ☐ $\mathcal{M} \vdash AF\neg a$
   - ☐ $\mathcal{M} \vdash E\neg Fa$
   - ☐ $\mathcal{M} \vdash EFa$

   Une structure de Kripke satisfait une formule LTL si tous les chemins initiaux satisfont cette formule. Soit la structure $\mathcal{M}$ ci-dessous :



- $\mathcal{M} \vdash Ga$
- $\mathcal{M} \vdash X(a \wedge b)$

Exercice 3 (9 points)

Imaginons un moteur de recherche sémantique des images qui fonctionne en mode recherche vocale. Ce moteur est doté d'une base de données d'images annotées respectivement à une ontologie: les images joueront le rôle des individus.

L'ontologie est formalisée en Logique de Description comme suit :

$C1 \sqsubseteq T$	$C4 \sqsubseteq C5$
$C2 \sqsubseteq C1$	$C5 \sqsubseteq C1$
$C3 \sqsubseteq C1$	$C6 \sqsubseteq C5$
$C3 \sqsubseteq \perp$	
$C6 \sqsubseteq \perp$	$C1(image1)$
$C4 \sqsubseteq \perp$	$C2(image2)$
$C4 \sqsubseteq C3$	$C5(image3)$

Ce moteur de recherche exploite des mesures de similarité pour répondre à une requête utilisateur. Pour simplifier, considérons que la requête consiste en un seul mot. Les résultats d'une recherche sont organisés ***par ordre décroissant de pertinence***.

1. Ce moteur de recherche peut être vu comme un agent intelligent: donner une spécification PEAS adéquate.
2. Quels sont les traitements du TALN exploités dans ce moteur de recherche ?
3. Supposons que l'utilisateur cherche le concept C3. Donner les résultats renvoyés suite à cette requête si la métrique basée sur la longueur de chemin est utilisée.
4. Même question si on cherche C6 et la métrique utilisée est celle de Resnik.
5. On veut vérifier si une image est déjà indexée par notre ontologie : quelle procédure doit-on mener pour ce faire ?