IFT-17587
Intelligence Artificielle II

Agents logiques

Plan

- Agents à base de connaissances
- Le monde du wumpus
- La logique
- La logique propositionnelle

Agents qui résonnent logiquement

- Agents logiques: agents basés sur les connaissances disponibles concernant le monde et un raisonnement (logique) portant sur les actions possibles dans ce monde.
- Les agents logiques doivent connaître :
  - L'état actuel du monde.
  - Comment le monde change dans le temps?
  - Qu'est ce qu'il faut accomplir?
  - Quelles sont les conséquences des actions dans différentes circonstances?

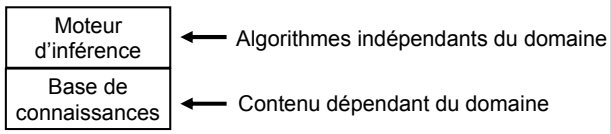
Agents à base de connaissances

- Base de connaissances: un ensemble de représentations de faits concernant le monde
  - chaque représentation est appelée une *phrase*
  - une base de connaissances est un ensemble de *phrases* exprimées dans un *langage formel*
  - les phrases sont exprimées dans un langage de représentation des connaissances.

Gestion des connaissances

- L'ajout de connaissances est symbolisé par l'action *Tell* et l'interrogation (requête) est symbolisée par l'action *Ask*.
- La réponse à une requête (*Ask*) doit découler de ce qui a été ajouté (*Tell*) dans la base de connaissances.
- La base de connaissances ne peut pas inventer, elle doit déduire (inférer) à partir de ses mécanismes de déduction (moteur d'inférence).
- La base de connaissances peut contenir des informations initiales, i.e. des connaissances de base (background knowledge).

Structure



Algorithme

- *ASK* raisonne logiquement afin de prouver quelle est la meilleure action, en accord avec les connaissances et les buts de l'agent.

```
function KB-AGENT(percept) returns an action
  static: KB, a knowledge base
          t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

- Le détail du mécanisme d'inférence est caché dans *Tell* et *Ask* qui sont l'interface avec le système de raisonnement.
- Make-Percept-Sentence, Make-Action-Query et Make-Action-Sentence ont comme valeur de retour une phrase dans le langage de représentation approprié. Elles cachent les détails du langage de représentation.

Capacités d'un agent

- Un agent à base de connaissances doit avoir les capacités suivantes:
  - représenter des états, des actions, etc.
  - incorporer de nouvelles perceptions
  - faire la mise à jour de sa représentation du monde
  - déduire des propriétés cachées du monde
  - déduire des actions appropriées

Description d'un agent

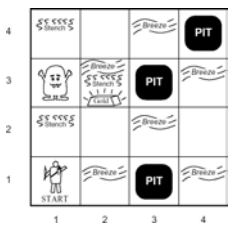
- Un agent à base de connaissances peut être décrit selon deux niveaux:
  - Niveau connaissances: description de l'agent par une description de ce qu'il sait.
  - Niveau implémentation: Description des structures de données de la base de connaissances et des algorithmes qui les manipulent.

Connaissances initiales

- Acquisition des connaissances initiales:
  - Approche déclarative: les connaissances initiales de l'agent sont ajoutées avec TELL, avant toutes perceptions.
  - Approche procédurales: les comportements désirés sont programmés directement.
- On peut aussi donner à l'agent la possibilité d'apprendre de nouvelles connaissances par lui-même pour obtenir un agent autonome.

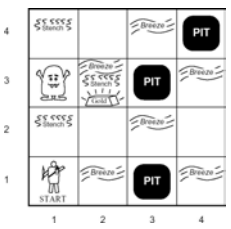
Le monde du *wumpus* (PEAS)

- Mesure de performance
  - +1000 pour l'or, -1000 pour tomber dans un trou, -1 pour chaque action, -10 pour tirer une flèche.
- Environnement
  - Une grille de 4 x 4
  - L'agent commence en [1,1] en regardant à droite
  - Les locations de l'or, du wumpus et des trous sont choisis aléatoirement.



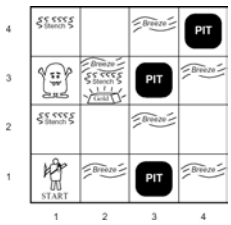
Le monde du *wumpus* (PEAS)

- Effecteurs
  - L'agent peut avancer, tourner à gauche ou tourner à droite.
  - L'agent meurt s'il tombe dans un trous ou arrive sur la même case que le wumpus
  - L'agent peut être sur la même case qu'un wumpus mort.
  - Avancer n'a aucun effet s'il y a un mur
  - L'action *Prendre* permet de ramasser un objet
  - L'action *Tirer* permet de tirer un flèche en avant si l'agent en a une.



Le monde du *wumpus* (PEAS)

- Capteurs
  - L'agent perçoit une puanteur sur la case du wumpus et sur les cases adjacentes.
  - L'agent perçoit une brise sur les cases adjacentes à un trou.
  - L'agent perçoit scintillement s'il est sur la case de l'or.
  - Si l'agent avance dans un mur, il va percevoir une collision.
  - Lorsque le wumpus meurt, l'agent va percevoir un cri



Propriété de l'environnement

- Observable ?
  - Non, uniquement une perception locale
- Déterministe ?
  - Oui, les effets des actions sont spécifiés exactement
- Épisodique ?
  - Non, c'est séquentiel au niveau des actions
- Statique ?
  - Oui, le wumpus et les trous ne bougent pas
- Discret ?
  - Oui
- Multiagent ?
  - Non, le wumpus n'est qu'une composante de l'environnement

Action et raisonnement

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2	3,2	4,2
1,1 A OK	2,1 OK	3,1	4,1

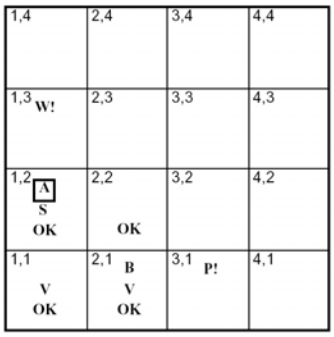
A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

Action et raisonnement

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2 P?	3,2	4,2
1,1 V OK	2,1 A B OK	3,1 P?	4,1

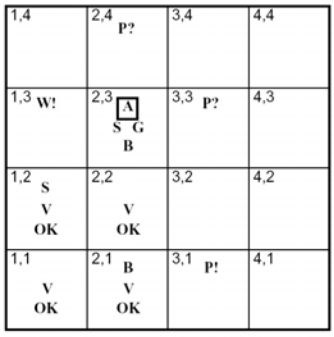
A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

Action et raisonnement



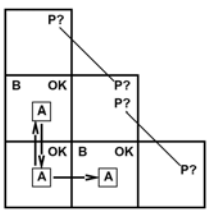
A = Agent
B = Breeze
G = Glitter, Gold
OK = Safe square
P = Pit
S = Stench
V = Visited
W = Wumpus

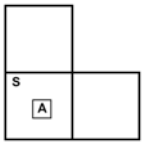
Action et raisonnement



A = Agent
B = Breeze
G = Glitter, Gold
OK = Safe square
P = Pit
S = Stench
V = Visited
W = Wumpus

Décisions difficiles





- Brise en (1,2) et (2,1)
  - Aucun déplacement sécuritaire
  - L'agent peut tenter sa chance
- Puanteur en (1,1)
  - Aucun déplacement sécuritaire
  - Stratégie: lancer une flèche
    - Si le wumpus était là, il est mort, donc c'est sécuritaire.
    - Si le wumpus n'était pas là, c'est sécuritaire.

La logique

- Les logiques sont des langages formels pour représenter de l'information de manière à permettre d'en déduire des conclusions.
- Syntaxe: Définit les configurations possibles pouvant constituer des phrases.
- Sémantique: Définit le sens d'une phrase, c'est-à-dire, définit la véracité d'une phrase.

La logique

- Exemple: le langage de l'arithmétique
  - « x + 2 > y » est une phrase, mais « x2 + y > » n'est pas une phrase.
  - « x + 2 > y » est vrai si le nombre x + 2 est plus grand que le nombre y.
  - « x + 2 > y » est vrai dans un monde où x = 7 et y = 1.
  - « x + 2 > y » est faux dans un monde où x = 0 et y = 6.

Inférence

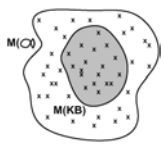
- La base de connaissance (KB) infère α si α est vrai dans tous les mondes où KB est vrai. On note cela

$$KB \models \alpha$$

- Exemple, x + y = 4 permet d'inférer que 4 = x + y

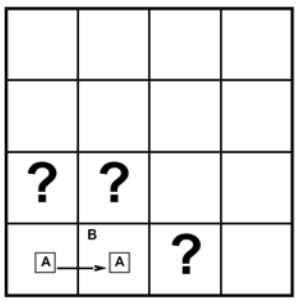
Modèles

- *m* est un modèle d'une phrase α si α est vrai dans *m*.
- M(α) est l'ensemble de tous les modèles de α.
- $KB \models \alpha$ ssi $M(KB) \subseteq M(\alpha)$
- Exemple,
  - KB = Canadiens ont gagnés et Sénateurs ont gagnés
  - α = Canadiens ont gagnés

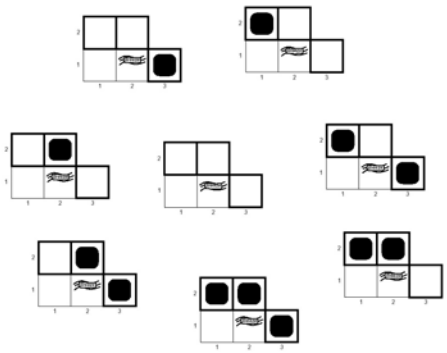


Exemple

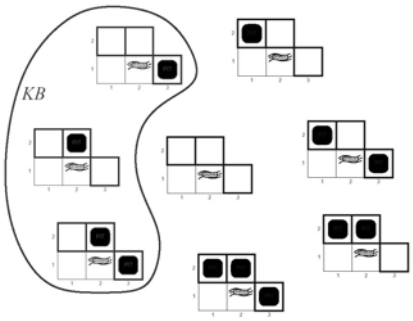
- Considérons les modèles possibles pour les trois « ? ».
- On ne s'attarde qu'au trous, donc il y a 2^3 modèles possibles.



Modèles possibles

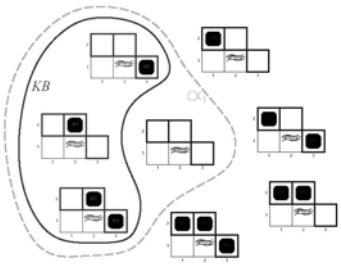


Modèles possibles



KB = Les règles du monde + les observations

Modèles possibles

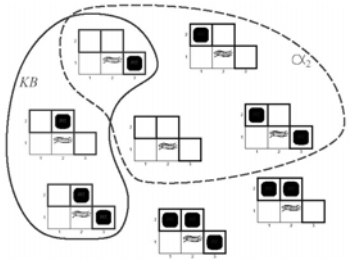


KB = Les règles du monde + les observations

$\alpha_1 =$ "(1,2) est OK", $KB \models \alpha_1$

prouvé par vérification de modèle

Modèles possibles



KB = Les règles du monde + les observations

$\alpha_2 =$ "(2,2) est OK", $KB \not\models \alpha_2$

Inférence

- Si un algorithme d'inférence *i* permet d'inférer α à partir de *KB*, on écrit:

$$KB \vdash_i \alpha$$

- Un algorithme d'inférence conserve la véracité (sound) si:

$$KB \vdash_i \alpha \Rightarrow KB \models \alpha$$

- Un algorithme d'inférence est complet si:

$$KB \models \alpha \Rightarrow KB \vdash_i \alpha$$

Remarques

- Il faut distinguer entre un fait et une phrase:
  - Un fait est une partie intégrante du monde.
  - Une phrase est une représentation encodée d'un fait qui peut être emmagasinée et manipulée par l'agent.
  - Les mécanismes de raisonnement opèrent sur les représentations des faits et non pas sur les faits eux-mêmes.

Remarques

- La procédure d'inférence:
  - génère de nouvelles phrases à partir de la base de connaissances
  - ou elle vérifie si une phrase peut-être dérivée à partir de la base de connaissances.
- Ce type de système préserve la vérité : à partir de phrases vraies dans la base de connaissances, d'autres phrases vraies sont générées par preuve.

Remarques

- Le système ne peut pas inférer des phrases qui contredisent la base de connaissances.

Logique propositionnelle

- C'est la logique la plus simple
- Les symboles propositionnels (P_1, P_2, etc.) sont des phrases.

5 connectifs

- Il y a 5 connectifs pour construire des phrases plus complexes:
  - Négation: si *P* est une phrases, alors ¬*P* est une phrase.
  - Conjonction: si P_1 *et* P_2 sont des phrases, alors $P_1 \wedge P_2$ est une phrase.
  - Disjonction: si P_1 *et* P_2 sont des phrases, alors $P_1 \vee P_2$ est une phrase.
  - Implication: si P_1 *et* P_2 sont des phrases, alors $P_1 \Rightarrow P_2$ est une phrase.
  - Biconditionnel: si P_1 *et* P_2 sont des phrases, alors $P_1 \Leftrightarrow P_2$ est une phrase.

Exemple du wumpus

- $P_{i,j}$ est vrai s'il y a un trou en (i,j)
- $B_{i,j}$ est vrai s'il y a une brise en (i,j)
- Exemple, il y a une brise dans les cases adjacentes à un trou
  - $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
  - $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
- Il y a une brise ssi il y a un trou dans une case adjacente.

Table de vérité pour l'inférence

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	KB
false	*false*	*false*	*false*	*false*	*false*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*false*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*false*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*

$O(2^n)$ pour n symboles.

Quelques concepts

- Deux phrases sont équivalente logiquement si elle sont vrais dans les mêmes modèles.
  - Ex: $A \wedge B$ est quivalent $B \wedge A$
- Une phrase est valide si elle est vrai dans tous les modèles.
  - Ex: $Vrai,\ A \vee \neg A,\ A \Rightarrow A$
  - La validité est reliée à l'inférence par le théorème de déduction:

$$KB \models \alpha \text{ ssi } (KB \Rightarrow \alpha) \text{ est valide}$$

Quelques concepts

- Une phrase est satisfiable si elle est vraie dans certains modèles.
  - Ex: $A \vee B$
- Une phrase est non satisfiable si elle est vrai dans aucun modèle.
  - Ex: $A \wedge \neg A$
- La satisfiabilité est reliée à l'inférence par:

$$KB \models \alpha \text{ ssi } (KB \wedge \neg\alpha) \text{ est non satisfiable}$$

C'est la preuve par contradiction

Preuve par résolution

- Si la phrase est sous forme normale conjonctive (CNF), on peut utiliser la règle d'inférence de résolution.
  - Exemple CNF: $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$
  - Règle de résolution: $\dfrac{l_1 \vee l_2,\quad \neg l_2 \vee l_3}{l_1 \vee l_3}$
  - Exemple: $\dfrac{P_{1,1} \vee P_{3,1},\quad \neg P_{1,1} \vee \neg P_{2,2}}{P_{3,1} \vee \neg P_{2,2}}$

Conversion en CNF

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

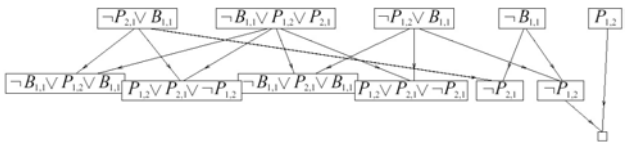
- Éliminer $\Leftrightarrow$, remplacer $\alpha \Leftrightarrow \beta$ par $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
  $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
- Éliminer $\Rightarrow$, remplacer $\alpha \Rightarrow \beta$ par $(\neg\alpha \vee \beta)$
  $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
- Déplacer les $\neg$ vers l'intérieur à l'aide des règles de de Morgan et de l'élimination de la double négation.
  $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
- Distribuer les $\vee$ et les $\wedge$ à l'aide de la règle de distributivité.
  $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

Algorithme de résolution

- Mettre la phrase $KB \wedge \neg\alpha$ sous forme normale conjonctive.
- C'est une preuve par contradiction.
- Appliquer la règle de résolution jusqu'à:
  - La règle n'est plus applicable, dans ce cas la clause α n'est pas prouvée.
  - On obtient la clause vide, donc α est prouvée.

Exemple de résolution

$KB = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$ $\alpha = \neg P_{1,2}$



Chaînage avant et arrière

- Il faut des clauses sous la forme de Horn
  - Une disjonction de littéraux avec un seul littéral positif:

$$(\neg L_{1,1} \vee \neg Brise \vee B_{1,1})$$

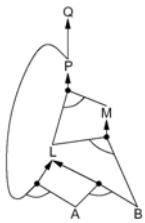
  - Une clause de Horn peut être représenté sous forme d'implication:

$$(L_{1,1} \wedge Brise) \Rightarrow B_{1,1})$$

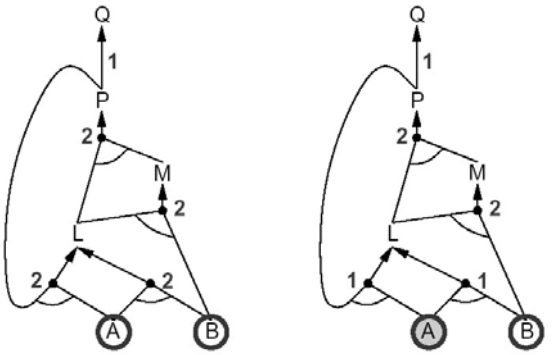
Chaînage avant

- Principe: appliquer toutes les règles applicable et ajouter leur conclusion à la base de connaissance jusqu'à ce qu'on trouve la requête.

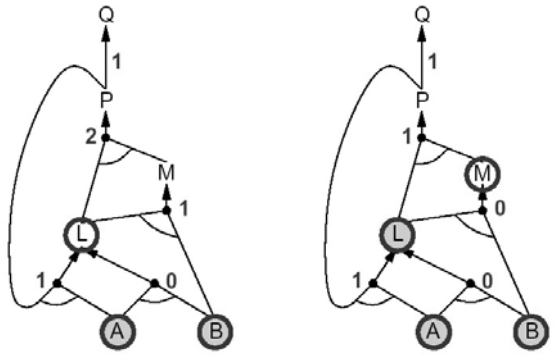
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



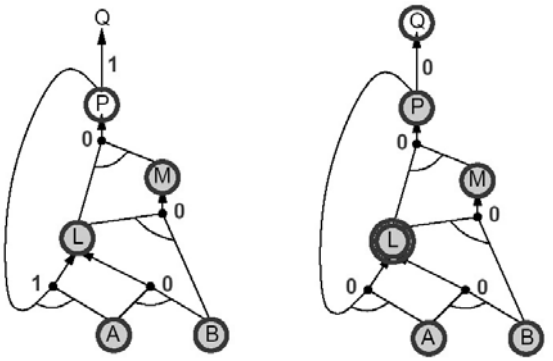
Chaînage en avant



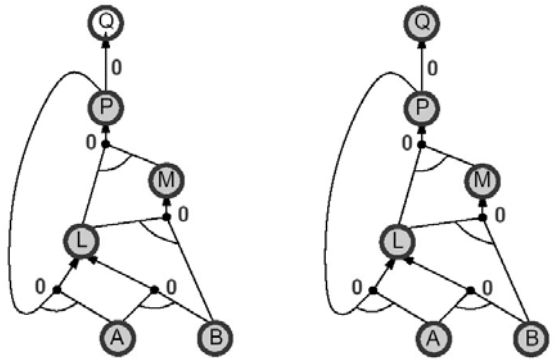
Chaînage en avant



Chaînage en avant



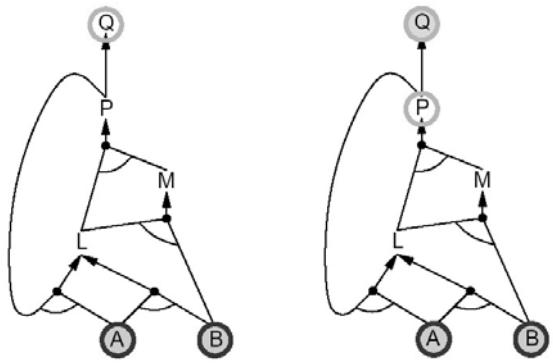
Chaînage en avant



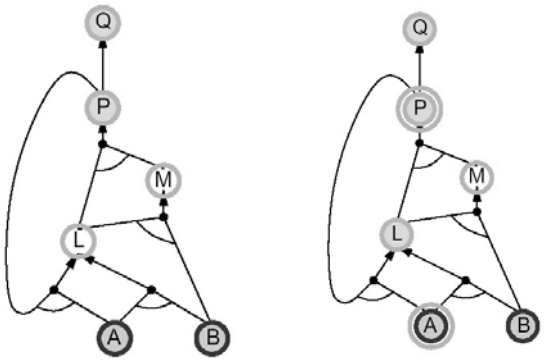
Chaînage arrière

- Principe: Travailler à l'envers à partir de la requête.
- Pour prouver *Q* par chaînage arrière:
  - Regarder si *Q* est connu, sinon
  - Prouver toutes les prémisses d'une règle ayant *Q* comme conclusion.
- Pour éviter les boucles: vérifier si un nouveau but est déjà sur la pile des buts
- Éviter le travail répété: vérifier si un nouveau but
  - a déjà été prouvé vrai, ou
  - a déjà échoué.

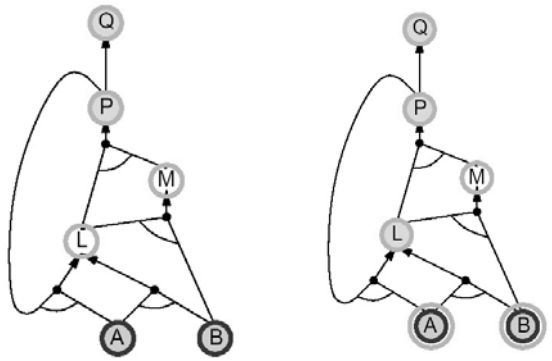
Chaînage arrière



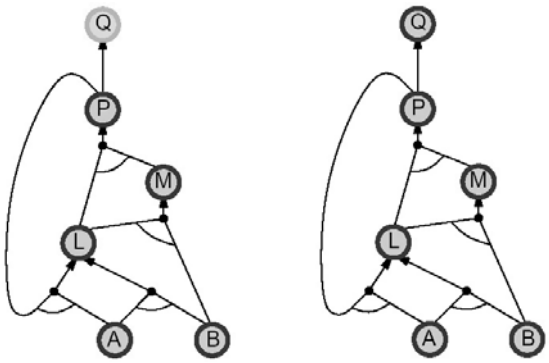
Chaînage arrière



Chaînage arrière



Chaînage arrière



Comparaison entre chaînage avant et arrière

- Le chaînage avant est dirigé par les données, c'est un processus automatique et inconscient.
  - Ex: Reconnaissance de formes
  - Peut effectuer du travail qui n'est pas utile pour la preuve.
- Le chaînage arrière est dirigé par le but. Il est utile pour les résolutions de problèmes.
  - Ex: Où sont mes clefs?

Remarques

- Les agents logiciels appliquent des règles d'inférence à leur base de connaissances pour inférer de nouvelles informations et prendre des décisions.
- Les recherches en avant et en arrière sont en temps linéaire, complète pour les clauses de Horn.
- La technique de résolution est en temps exponentiel, mais elle est complète pour la logique propositionnelle.
- La logique propositionnelle manque de pouvoir d'expression.

IFT-17587
Intelligence Artificielle II

Logique du premier ordre

Plan

- Les avantages et inconvénients de la logique propositionnelle.
- La logique du premier ordre (LPO).
- Interagir avec une base de connaissance (BC) en LPO.

Les avantages de la logique propositionnelle

- Elle est déclarative: les connaissances et le processus d'inférence sont séparés et l'inférence est indépendante du domaine.
- Permet d'avoir des informations partielles, disjointes et négatives.
- Elle est compositionnelle: la signification d'une phrase dépend de la signification de ses parties.
- La signification d'une phrase est indépendante du contexte.

Le gros désavantage de la logique propositionnelle

- Elle a un pouvoir de représentation très limité
  - Ex: On ne peut pas dire que les trous causes une brise dans les cases adjacentes.
  - Sauf, si on écrit une phrase pour chaque case.

Logique du premier ordre (LPO)

- Le monde est composé d'objets.
  - Ex: Gens, maisons, nombres, couleurs, etc.
- Il existe des relations entre les objets:
  - Unaire: ce sont des propriétés: rouge, rond, grand, petit.
  - N-aire: frère de, plus grand que, etc.
- Certaines de ces relations sont des fonctions: Une fonction est une relation où il n'y a qu'une seule valeur pour une entrée donnée.

Comparaison des différentes logiques

Language	Ontological Commitment	Epistemological Commitment
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief $\in [0,1]$
Fuzzy logic	degree of truth $\in [0,1]$	known interval value

Qu'est-ce qui existe dans le monde ?

Qu'est-ce que l'agent croit à propos des faits ?

Syntaxe de la LPO

- Éléments de base:
  - Constantes: Richard, Jean, 2, etc.
  - Prédicats: Frère, PlusGrandQue, etc.
  - Fonctions: RacineCarrée, JambeGaucheDe, etc.
  - Variables: x, y, z, etc.
  - Connecteurs: $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\Leftrightarrow$, etc.
  - Égalité: =
  - Quantificateurs: $\forall$, $\exists$

Phrase atomique

- Elle permet d'énoncer un fait.
- Ex: Frère(Richard, Jean), EstMarrié(PèreDe(Richard), MèreDe(Jean))
- Une phrase atomique est vraie si la relation référée par le symbole de prédicat tient entre les objets référés par les arguments.
- La véracité de la phrase dépend donc de son interprétation et du monde.

Phrase complexe

- On utilise des connecteurs logiques pour avoir des phrases plus complexes.
- Exemples:
  - $Frère(Richard, Jean) \wedge Frère(Jean, Richard)$
  - $Roi(Richard) \vee Roi(Jean)$
  - $\neg Roi(Richard) \Rightarrow Roi(Jean)$

Quantificateurs

- Un quantificateur permet d'exprimer des propriétés à propos d'une collection d'objets sans avoir à énumérer tous les objets par leur nom.
- La LPO a deux quantificateurs standards: universel et existentiel.

Quantificateur universel

- Vrai ssi toutes les phrases sont vraies.
- $\forall x P$ est vrai si *P* est vrai pour tous les objets x dans l'univers. D'où le nom de quantificateur universel.
- Tous les chats sont des mammifères:
  - $\forall x \ Chat(x) \Rightarrow Mammifère(x)$
- Tous dans la classe sont intelligents:
  - $\forall x \ Dans(x, Classe) \Rightarrow Intelligent(x)$
- Erreur courante:
  - $\forall x \ Dans(x, Classe) \wedge Intelligent(x)$
  - Tous sont dans la classe et tous sont intelligents

Quantificateur existentiel

- Vrai si certains des énoncés sont vrais.
- $\exists x P$ est vrai si *P* est vrai pour certains des objets dans l'univers.
- Spot a une sœur qui est un chat:
  - $\exists x \ Soeur(x, Spot) \wedge Chat(x)$
- Quelqu'un dans la classe est intelligent:
  - $\exists x \ Dans(x, Classe) \wedge Intelligent(x)$
- Erreur courante:
  - $\exists x \ Dans(x, Classe) \Rightarrow Intelligent(x)$
  - Est vraie si quelqu'un n'est pas dans la classe, ce qui ne dit pas grand chose.

Propriétés des quantificateurs

- $\forall x\ \forall y$ est la même chose que $\forall y\ \forall x$
- $\exists x\ \exists y$ est la même chose que $\exists y\ \exists x$
- $\exists x\ \forall y$ n'est pas la même chose que $\forall y\ \exists x$
  - $\exists x\ \forall y\ \ Aimer(x, y)$
  - Il existe une personne qui aime tout le monde.
  - $\forall y\ \exists x\ \ Aimer(x, y)$
  - Toute personne est aimé par au moins une personne.

Propriétés des quantificateurs

- Un quantificateur peut être exprimé en utilisant l'autre.
  - $\forall x\ \ P \equiv \neg\exists x\ \neg P$
  - $\exists x\ \ P \equiv \neg\forall x\ \neg P$
- Exemples:
  - $\forall x\ \ Aime(x, CremeGlacee) \equiv \neg\exists x\ \neg Aime(x, CremeGlacee)$
  - $\exists x\ \ Aime(x, Brocoli) \equiv \neg\forall x\ \neg Aime(x, Brocoli)$

Égalité

- Vrai si l'énoncé fait référence au même objet.
- Exemples:
  - Le père de Jean est Henry
    $Père(Jean) = Henry$
  - Richard a au moins deux frères
    $\exists x, y\ \ Frère(x, Richard) \wedge Frère(y, Richard) \wedge \neg(x = y)$
- On peut aussi utiliser la notation $x \neq y$ comme abréviation de $\neg(x = y)$

Interagir avec une BC en LPO

- Supposons un agent dans le monde du wumpus utilisant une BC en LPO.
  - L'agent perçoit une puanteur, une brise, mais pas de scintillement au temps 5.

$Tell(BC, Percept([Puanteur, Brise, Rien], 5))$

$Ask(BC, \exists a\ \ Action(a, 5))$

  - Réponse: Oui, {a / Tire} (Substitution)
- Exemple: $S = Soeur(x, y)$
  $\alpha = \{x/Marie, y/Jean\}$
  $S\alpha = Soeur(Marie, Jean)$

BC pour le monde du wumpus

- Perceptions
  $\forall b, s, t \ Percept([Puanteur, b, s], t) \Rightarrow Puanteur(t)$
  $\forall p, b, t \ Percept([p, b, Scintillement], t) \Rightarrow AtOr(t)$
- Réflexe
  $\forall t \ AtOr(t) \Rightarrow Action(Prendre, t)$
- Réflexe avec état interne
  $\forall t \ AtOr(t) \wedge \neg Possède(Or, t) \Rightarrow Action(Prendre, t)$
- On ne peut pas observer *Possède(Or,t)*, donc c'est important de tenir à jour les changements de l'environnement.

Définir l'environnement

- Propriétés des emplacements
  $\forall x, t \ At(Agent, x, t) \wedge Puanteur(t) \Rightarrow Pue(x)$
  $\forall x, t \ At(Agent, x, t) \wedge Venteux(t) \Rightarrow Brise(x)$
- Il y a une brise sur les cases adjacentes à un trou
  - Règle de diagnostic: inférer la cause à partir de l'effet
    $\forall y \ Brise(y) \Rightarrow \exists x \ Trou(x) \wedge Adjacent(x, y)$
  - Règle causal: inférer l'effet à par de la cause
    $\forall x, y \ Trou(x) \wedge Adjacent(x, y) \Rightarrow Brise(y)$
- Ces définitions ne sont pas complètes, la bonne définition est:
  $\forall y \ Brise(y) \Leftrightarrow [\exists x \ Trou(x) \wedge Adjacent(x, y)]$

IFT-17587
Intelligence Artificielle II

Inférence en logique du premier ordre

Plan

- Réduire l'inférence en LPO en inférence en logique propositionnelle
- Unification
- Modus Ponens Généralisé
- Chaînage avant
- Chaînage arrière
- Résolution

Inférence pour la LPO

- L'inférence est utilisée comme processus de raisonnement.
- On utilise les connaissances et l'inférence pour construire un programme qui raisonne.
- Inférence: trouver α tel que $KB \models \alpha$
  - C'est-à-dire, montrer que α peut être dérivé de la base de connaissances.
- Une preuve est un processus de recherche, les opérateurs sont les règles d'inférence.

Règles d'inférence

- Modus Ponens: (Implication-Élimination) On peut inférer la conclusion à partir de l'implication et de la prémisse.

$$\frac{\alpha \Rightarrow \beta,\ \alpha}{\beta} \qquad \frac{Est(Joe, UL) \Rightarrow OK(Joe),\ Est(Joe, UL)}{OK(Joe)}$$

- And-Elimination: On peut inférer une des phrases à partir de sa conjonction avec d'autres.

$$\frac{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}{\alpha_i} \qquad \frac{OK(Joe) \wedge EnInfo(Lucie)}{EnInfo(Lucie)}$$

Règles d'inférence

- And-Introduction: On peut inférer la conjonction à partir d'une liste de phrases.

$$\frac{\alpha_1, \alpha_2, \ldots, \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n} \qquad \frac{OK(Joe), EnInfo(Lucie)}{OK(Joe) \wedge EnInfo(Lucie)}$$

- Or-Introduction: On peut inférer la disjonction d'une phrase avec d'autres.

$$\frac{\alpha_i}{\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n} \qquad \frac{OK(Joe)}{\ldots \vee OK(Joe) \vee \ldots}$$

Règles d'inférence

- Double-Negation Elimination: On peut inférer une phrase vraie de sa double négation.

$$\frac{\neg\neg\alpha}{\alpha} \qquad \frac{\neg\neg OK(Joe)}{OK(Joe)}$$

- Unit-Resolution: On peut inférer qu'une phrase est vraie à partir de sa disjonction avec une phrase fausse.

$$\frac{\alpha \vee \beta, \neg\beta}{\alpha} \qquad \frac{OK(Joe) \vee EnInfo(Lucie), \neg OK(Joe)}{EnInfo(Lucie)}$$

Règles d'inférence

- Resolution: L'implication est transitive.

$$\frac{\alpha \vee \beta, \neg\beta \vee \gamma}{\alpha \vee \gamma} \qquad \frac{\neg\alpha \Rightarrow \beta, \beta \Rightarrow \gamma}{\neg\alpha \Rightarrow \gamma}$$

Instanciation universelle

- On peut substituer une variable par une constante.

$$\frac{\forall x\ \alpha}{SUBST(\{v/C\}, \alpha)} \qquad \frac{\forall x\ Est(x, UL) \Rightarrow OK(x)}{Est(Pat, UL) \Rightarrow OK(Pat)}$$

- Autre exemple:

$$\forall x\ Aime(x, CrèmeGlacée)$$

– Et la substitution {x/Joe}, on peut inférer

$$Aime(Joe, CrèmeGlacée)$$

Instanciation existentielle

- Pour toute phrase α, variable *x* et constante *c* (constante de Skolem) qui n'apparaît nulle par ailleurs dans la base de connaissances:

$$\frac{\exists x \ \alpha}{SUBST(\{x/C\}, \alpha)}$$

- Autre exemple:

$$\exists x \ Aime(x, CrèmeGlacée)$$

  - On peut inférer $Aime(C_1, CrèmeGlacée)$
  - À condition que C_1 ne soit pas dans la base de connaissances.

Remarques sur l'instanciation

- L'instanciation universelle peut être appliquer plusieurs fois pour ajouter de nouvelles phrases.
  - La nouvelle BC est logiquement équivalente à l'ancienne.
- L'instanciation existentielle ne peut être appliquer qu'une fois pour remplacer une phrase existentielle.
  - La nouvelle BC n'est pas équivalente à l'ancienne, mais elle est satisfiable si l'ancienne était satisfiable.

Réduction à une inférence propositionnelle

- Supposons que la base de connaissances ne contient que:

$$\forall x \ Roi(x) \wedge Avide(x) \Rightarrow Malveillant(x)$$
$$Roi(Jean)$$
$$Avide(Jean)$$
$$Frère(Richard, Jean)$$

- Si on instancie la phrase universelle de toutes les manières possibles:

$$Roi(Jean) \wedge Avide(Jean) \Rightarrow Malveillant(Jean)$$
$$Roi(Richard) \wedge Avide(Richard) \Rightarrow Malveillant(Richard)$$
$$Roi(Jean)$$
$$Avide(Jean)$$
$$Frère(Richard, Jean)$$

Méthode

- On peut transformer la BC sous forme propositionnelle et appliquer la technique de résolution pour obtenir la réponse.
  - Cette méthode est complète, i.e. toutes les phrases dérivables peuvent être prouvées.
- Le problème, c'est que la méthode est semi-décidable:
  - L'algorithme peut toujours dire oui, si la phrase est dérivable.
  - Mais, l'algorithme boucle à l'infini sinon.

Autre problème

- En rendant la BC sous forme propositionnelle, on ajoute des phrases non pertinentes.

$$\forall x\ Roi(x) \wedge Avide(x) \Rightarrow Malveillant(x)$$
$$Roi(Jean)$$
$$\forall y\ Avide(y)$$
$$Frère(Richard, Jean)$$

- Il est évident que *Malveillant(Jean)*, mais on produit aussi beaucoup de faits non pertinents comme *Avide(Richard)*.

Modus Ponens Généralisé

- Si on a des phrases atomiques p_i, p_i' et q, et qu'il existe une substitution theta tel que SUBST(theta, p_i') = SUBST(theta, p_i), pour tout *i*, alors

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{SUBST(\theta, q)}$$

- Exemple:

p_1' est $Roi(Jean)$
p_2' est $Avide(y)$
θ est $\{x/Jean, y/Jean\}$
$SUBST(\theta, q)$ est $Malveillant(Jean)$

p_1 est $Roi(x)$
p_2 est $Avide(x)$
q est $Malveillant(x)$

Unification

- Pour trouver l'inférence immédiatement, il a fallu trouver la substitution qui unifiait Roi(x), Avide(x) avec Roi(Jean), Avide(y).
  - La substitution suivante fonctionne:

$$\theta = \{x/Jean, y/Jean\}$$

- L'opérateur d'unification est défini de la manière suivante:

$UNIFY(p, q) = \theta$ si $SUBST(\theta, p) = SUBST(\theta, q)$

Exemple d'unification

p	q	UNIFY(p,q)
Ami(Jean,x)	Ami(Jean,Jeanne)	{x/Jeanne}
Ami(Jean,x)	Ami(y,Julie)	{x/Julie, y/Jean}
Ami(Jean,x)	Ami(y,Mère(y))	{y/Jean, x/Mère(Jean)}
Ami(Jean,x)	Ami(x,Julie)	Échoue

« Standardizing apart » permet d'éviter les conflits entre variables, en renommant une variable.

$$UNIFY(Ami(Jean, x), Ami(z, Julie)) = \{x/Julie, z/Jean\}$$

L'unificateur le plus général

- Celui qui a le moins de contraintes sur les valeurs des variables.
  - Ex: UNIFY(Ami(Jean, x), Ami(y,z)) peut retourner {y/Jean, x/z} ou {y/Jean, x/Jean, z/Jean}
  - Le premier va donc donner Ami(Jean,z) et le deuxième Ami(Jean,Jean)
  - Le premier est donc plus général que le 2e
- Pour une paire d'expressions, il n'y a qu'un seul unificateur le plus général.

Chaînage avant et arrière

- Les règles doivent être sous la forme de clauses définies, i.e.
  - Une phrase atomique, ex: Roi(Jean)
  - ou une implication où la prémisse est une conjonction de littéraux positifs et que la conclusion est un seul littéral positif.

$$Roi(x) \wedge Avide(x) \Rightarrow Malveillant(x)$$

- Les variables sont supposées universellement quantifiées.

Exemple de BC

The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

Prove that Col. West is a criminal

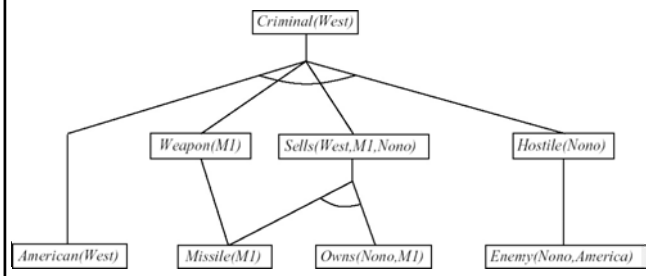
Exemple de BC

... it is a crime for an American to sell weapons to hostile nations:
$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$
Nono ... has some missiles, i.e., $\exists x\ Owns(Nono, x) \wedge Missile(x)$:
$Owns(Nono, M_1)$ and $Missile(M_1)$
... all of its missiles were sold to it by Colonel West
$\forall x\ Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
Missiles are weapons:
$Missile(x) \Rightarrow Weapon(x)$
An enemy of America counts as "hostile":
$Enemy(x, America) \Rightarrow Hostile(x)$
West, who is American ...
$American(West)$
The country Nono, an enemy of America ...
$Enemy(Nono, America)$

Chaînage avant

- Générer les conséquences à partir des faits de la BC.
- Tant que c'est possible, si toutes les prémisses d'une règle sont vraies, ajouter la conséquence de la règle à la BC.

Exemple chaînage avant



Propriétés du chaînage avant

- Valide et complet pour des clauses définies de premier ordre.
- Datalog: juste des clauses définies de premier ordre, sans fonction.
  - Le chaînage avant est assuré de terminer.
- S'il y a des fonctions, l'algorithme peut boucler à l'infini lorsque la question n'est pas dérivable.

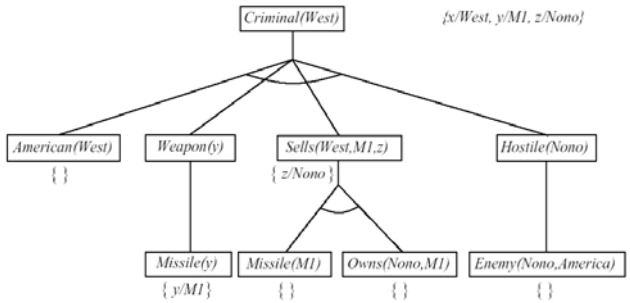
Remarques chaînage avant

- Observation: On n'a pas besoin d'essayer d'unifier une règle à l'itération *k*, si elle ne contient pas de prémisses ajoutées à l'itération *k* -1.
- Dirigée par les données: construction graduelle de la situation à mesure que les données sont ajoutées.
- Processus d'inférence qui ne tente pas de résoudre un problème.
- Processus qui manque de guidage et qui peut générer beaucoup de conclusions inutiles.

Recherche à retour arrière

- À partir de quelque chose que l'on veut prouver, trouver les implications qui permettent de le conclure et prouver leurs prémisses.
- Utilise Modus Ponens à l'envers.
- Utilisé lorsque l'on veut prouver quelque chose.
- C'est la fonction *ASK* de l'agent à base de connaissances.

Exemple de recherche à retour arrière



Propriétés de recherche à retour arrière

- C'est une recherche récursive en profondeur d'abord.
- C'est une méthode incomplète, parce qu'il peut y avoir des boucles.
  - Fixé en vérifiant le but courant avec les buts sur la pile des buts.
- Inefficace, parce qu'il peut y avoir des sous-buts qui se répètent.
  - Fixé en mémorisant les résultats passés et en les appliquant de nouveau.
- Elle est à la base des langages logique comme Prolog.

Résolution

- La règle de résolution est:

$$\frac{l_1 \vee \ldots \vee l_k, \;\; m_1 \vee \ldots \vee m_n}{SUBST(\theta, l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)}$$

  - Où $UNIFY(l_i, \neg m_j) = \theta$
- Exemple:

$$\frac{\neg Riche(x) \vee NonHeureux(x), \;\; Riche(Ken)}{NonHeureux(Ken)}$$

  - Avec: $\theta = \{x/Ken\}$
- Appliquer la résolution à: $CNF(BC \wedge \neg\alpha)$
- Complète pour la LPO.

Convertir en CNF

- Exemple: Tout le monde qui aime tous les animaux est aimé par quelqu'un:

$$\forall x\ [\forall y\ Animal(y) \Rightarrow Aime(x,y)] \Rightarrow [\exists y\ Aime(y,x)]$$

- Éliminer les biconditionnelles et les implications:

$$\forall x\ [\neg\forall y\ \neg Animal(y) \vee Aime(x,y)] \vee [\exists y\ Aime(y,x)]$$

- Déplacer les négations à l'intérieur:

$$\forall x\ [\exists y\ Animal(y) \wedge \neg Aime(x,y)] \vee [\exists y\ Aime(y,x)]$$

Convertir en CNF

- Standardiser les variables (une par quantificateur):

$$\forall x\ [\exists y\ Animal(y) \wedge \neg Aime(x,y)] \vee [\exists z\ Aime(z,x)]$$

- Skolemisation, les variables existentielles sont remplacées par des fonctions de Skolem:

$$\forall x\ [Animal(F(x)) \wedge \neg Aime(x,F(x))] \vee Aime(G(x),x)$$

- Enlever les quantificateurs universelles:

$$[Animal(F(x)) \wedge \neg Aime(x,F(x))] \vee Aime(G(x),x)$$

- Distribué les et:

$$[Animal(F(x)) \vee Aime(G(x),x)] \wedge$$
$$[\neg Aime(x,F(x)) \vee Aime(G(x),x)]$$

Exemple de résolution