Bases de l'intelligence artificielle
Programmation Logique

PR. M. THIAM
MAITRE DE CONFERENCES INFORMATIQUE
UFR SET – UT, 2016/2017

From scratch …

Résoudre un problème	**Indiquer où vous habitez**
Décrire le cheminement permettant d'arriver à la solution d'un problème décrit par un énoncé	Décrire le cheminement à emprunter pour arriver à votre habitation telle que décrit par votre adresse

?

Quelques problèmes

- Un nombre N est-il divisible par 4 ?

- Soit une série de nombre, trier ces nombres

- Soit le problème classique de la tour de Hanoi, afficher la liste des mouvements nécessaires pour le résoudre.

- Un voyageur de commerce désire faire sa tournée, existe-t-il une tournée de moins de 50 km ?

Équation diophantienne

- En arithmétique : Une équation diophantienne, en mathématiques, est une équation dont les
  - coefficients sont des nombres entiers et dont
  - solutions recherchées sont également entières.
- Le terme est aussi utilisé pour les équations à coefficients rationnels.

David Hilbert
& son problème n°10

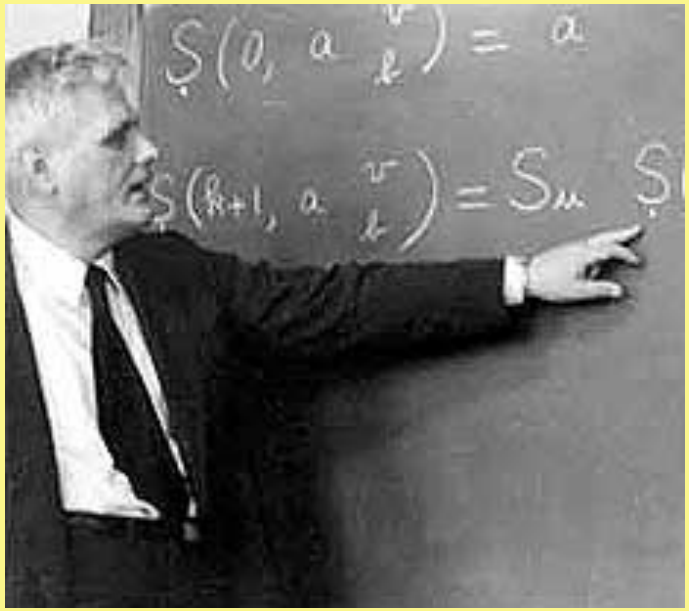
- 1862 - 1943
- Liste des 23 problèmes de Hilbert (1900)
- Problème numéro 10 :

« Trouver un algorithme déterminant si une équation diophantienne à des solutions »

Équation diophantienne

- Carl Friedrich Gauss, un mathématicien du XIX^{e} siècle, disait : « Leur charme particulier vient de la simplicité des énoncés jointe à la difficulté des preuves »
- Identité de **Bézout :** $a\,x + b\,y = c$
- Théorème de **Wilson** : $(x-1)! + 1 = y\,x$
- Triplet **Pythagoricien** : $x^2 + y^2 = z^2$
- Dernier théorème **Fermat** : $(n=4) : x^4 + y^4 = z^4$

Alonzo Church & le λ-calcul



- 1903 - 1995
- Résultat sur la calculabilité
- Développement du lambda-calcul
- 1936 : Démontre l'existence d 'un problème indécidable
- Thèse de Church

Kurt Godël

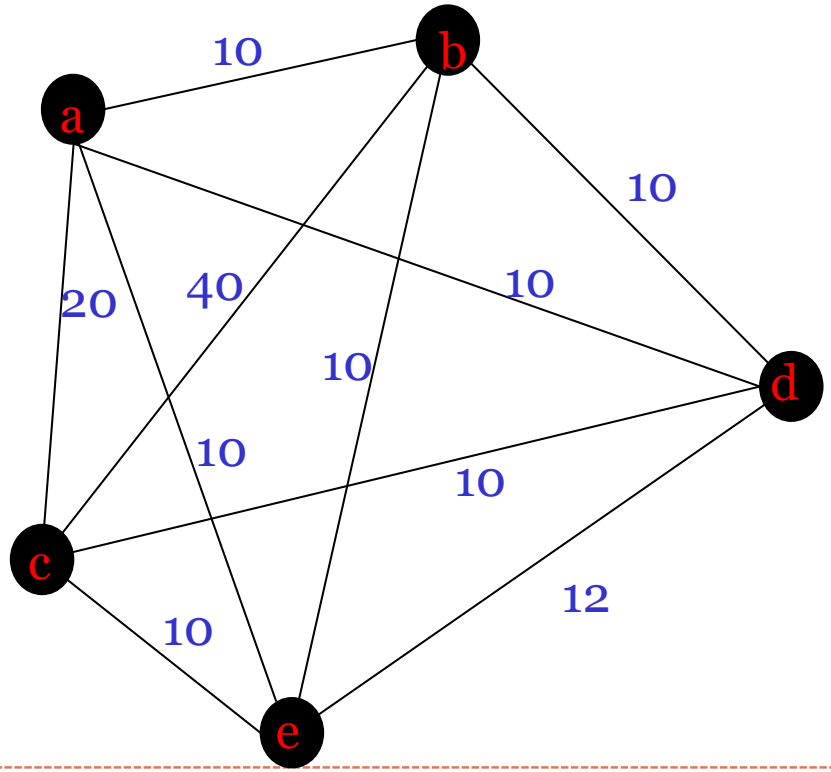
- 1906 - 1978
- Théorème d ’Incomplétude

« pour tout système formel S contenant le langage de l'arithmétique, il existe une proposition G indémontrable dans S »

Exemple de problème de NP

Le voyageur de commerce

• Un voyageur de commerce désire faire sa tournée, existe-t-il une tournée de moins de 50 km ?

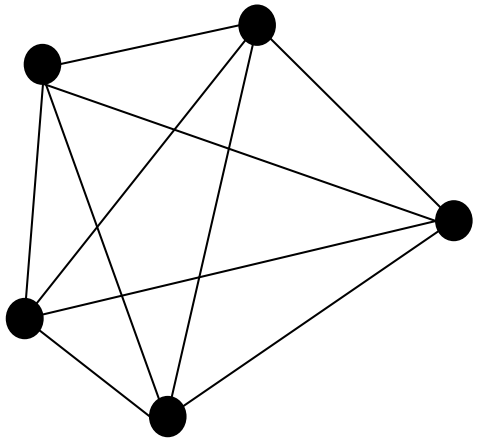


a-b-e-c-d-a	50km

NP et co-NP ?

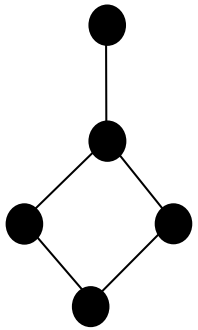
Problème du circuit hamiltonien

Un graphe est-il hamiltonien ?



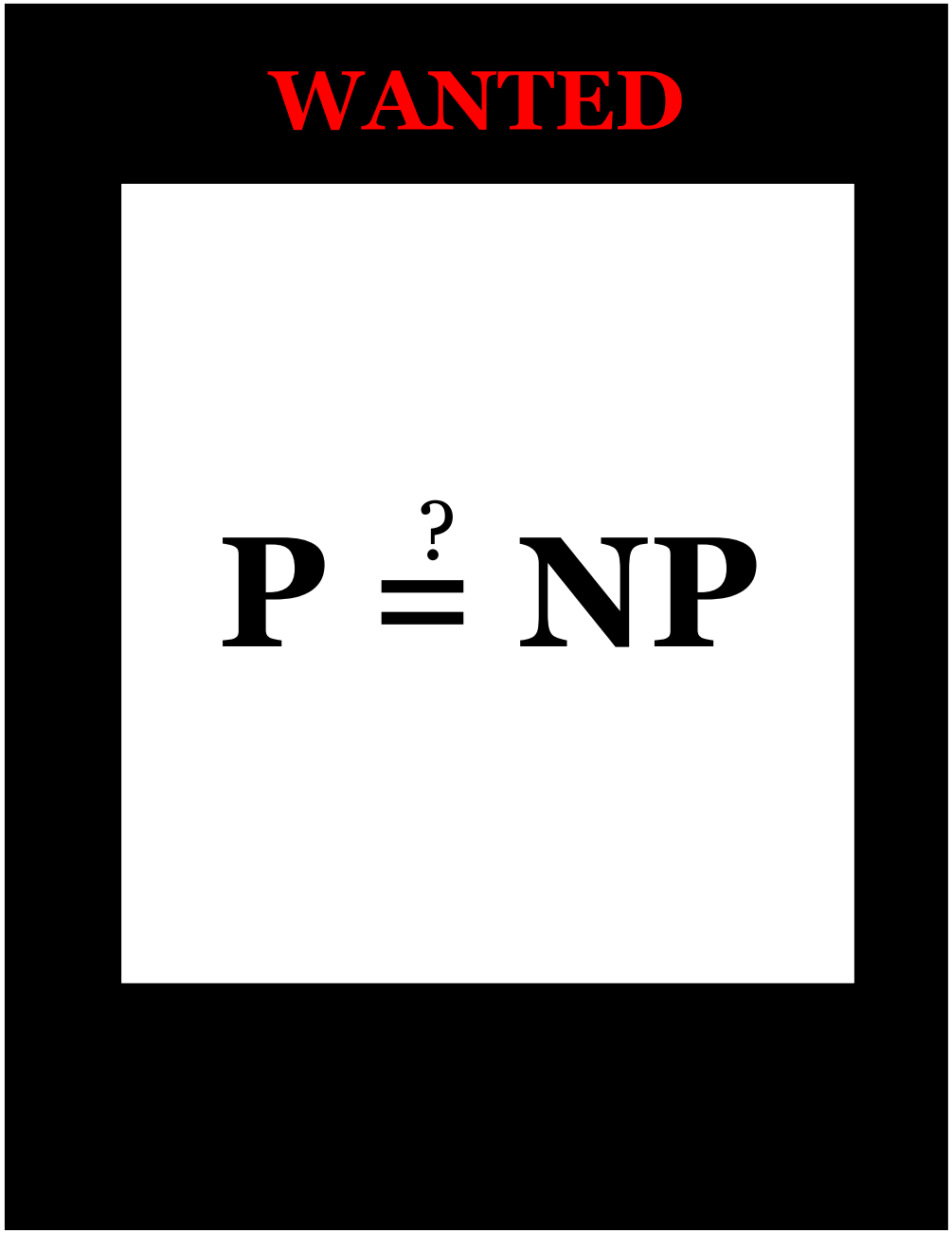
Ce problème ∈ NP

Un graphe n'est-il pas hamiltonien ?



Ce problème ∈ co-NP

Est-ce que ce problème ∈ NP ?


WANTED
P ≟ NP

Des algorithmes …

- Résous le toi-même
- Cherchez dans la ville
- Lire la notice
- Faites de sorte que ca marche
- Etc …

Introduction (1)

- Les techniques que nous connaissons ne permettent pas de simuler le comportement d'une personne, par exemple :
  - apprendre à utiliser un logiciel en lisant le manuel
  - s'apercevoir qu'on a été insulté
  - rédiger une dissertation sur un sujet donné
  - traduire un texte d'une langue A → B

Introduction (2)

- Pour cela il faut des objets qui relient les éléments linguistiques à des représentations du monde
- Des objets qui représentent la sémantique du texte

Introduction (3)

- Déterminer si des énoncés sont VRAIS / FAUX
  - Si Superman voulait et pouvait prévenir le mal, il le ferait.
  - Si Superman ne pouvait prévenir le mal, il serait impuissant;
  - S'il ne voulait pas prévenir le mal, il serait malveillant.
  - Superman ne prévient pas le mal.
  - Si Superman existe, il n'est ni impuissant, ni malveillant.

Introduction (4)

- Déterminer si des énoncés sont VRAIS / FAUX
  - Par conséquent, Superman n'existe pas.
  - v est dans le tableau **b**[1..10]
  - si la valeur **v** est dans **b**,
  - alors elle n'est pas dans **b**[11..20]

Expressions

- Constantes
  - 42
- Variables
  - X
- Opérateurs
  - ÷, +, −, <, =, .
- Exemples
  - 13, z, x+3+z2, x+5>8÷x2

Chapitre 1

LOGIQUE PROPOSITIONNELLE

Présentation

- Le ***calcul des propositions*** ou ***calcul propositionnel*** est une théorie logique qui définit les lois formelles du raisonnement.
- La version moderne de la logique stoïcienne.
- La première étape dans la construction des outils de la logique mathématique

Proposition

- **Définition**
  - Proposition = construction syntaxique pour laquelle il est sensé de parler de vérité.
- **Exemples**
  - « 2 + 2 = 4 »
  - « le livre est ouvert »
- Contre-exemples
  - « Que Dieu nous protège ! »
  - « viens ! »
  - « tais-toi ! »

Calcul des propositions

- 1ère étape dans la définition de la logique et du raisonnement
- Définition des règles de déduction qui relient les propositions entre elles (sans en examiner le contenu)
- 1ère étape dans la construction du calcul des prédicats
  - s'intéresse au contenu des propositions et qui est une formalisation achevée du raisonnement mathématique.

Calcul ou système déductif

- Ensemble de règles permettant en un nombre fini d'étapes et selon des règles explicites de déterminer si une proposition est vraie.
- Procédé s'appelle une *démonstration*.
- Structure mathématique permet de garantir que ces raisonnements ou démonstrations ont du sens (sémantique)
- Sémantique → 2 valeurs, *vrai* et *faux* (notés 1 et 0).

Syntaxe du langage

- **variables propositionnelles** ou **propositions atomiques**, notées p, q, etc.
- **opérateurs** ou **connecteurs**.

$\equiv, =, \Leftrightarrow$	égalité, équivalence
$\not\equiv, \neq$	inégalité, inéquivalence
$\neg$	négation, non
$\vee$	disjonction, ou
$\wedge$	conjonction, et
$\Rightarrow$	implication, si . . .alors
$\Leftarrow$	conséquence

Autres symboles

- Constante noté ⊥,
  - prononcé taquet vers le haut, type vide, bottom ou bot, qui vise à représenter le *faux*.
- () parenthèses pour lever les ambiguïtés dans les formules
- Formules propositionnelles

Formules propositionnelles

- Règles de formation indiquent
  - comment construire des propositions complexes à partir des propositions élémentaires, ou atomes (variables propositionnelles, constantes comme $\perp$).
  - quels assemblages de signes, quels mots, sont bien formés et désignent des propositions.
- Définition dépend d'un choix de connecteurs, et d'un choix d'atomes.

Exemples de formules

- P un ensemble de variables propositionnelles
  - $p \in P$ est une formule
  - $\bot$ est une formule
  - si p et q sont des formules, alors $(p \wedge p)$, $(p \vee q)$, $(p \rightarrow q)$, $(p \leftrightarrow q)$ et $\neg p$ sont des propositions

Systèmes déductifs

- **La déduction à la Hilbert**
  - Modus penens
- **La déduction naturelle**
- **Le calcul des séquents**

Egalité

- Si X et Y ont la même valeur alors
  - (X=Y)=Vrai;
- Si X et Y n'ont pas la même valeur alors
  - (X=Y)=Faux;
- Raisonner au sujet de l'égalité **X = Y** sans toujours devoir évaluer X et Y
  - (X=Y) = (Y=X)

Lois de l'égalité

- Réflexivité
  - x=x
- Symétrie (commutativité)
  - (x=y) = (y=x)
- Transitivité
  - x=y, y=z => x=z

Lois de l'égalité

- Loi de **Leibniz** (~350 ans)

  *Deux expressions **A** et **B** sont égales si et seulement si le remplacement de l'une par l'autre dans n'importe quelle expression **E** ne change pas la valeur de **E**.*

Tables de vérités

		OR	NOR	AND	NAND
F	F	F	V	F	V
F	V	V	F	F	V
V	F	V	F	F	V
V	V	V	F	V	F

		XOR	NXOR	→	¬
F	F	F	V	V	V
F	V	V	F	V	V
V	F	V	F	F	F
V	V	F	V	V	F

Exemples

- j'ai une auto rouge $\vee$ j'étudie l'informatique
- j'ai une auto rouge OUEX j'étudie l'informatique
- $x>2 \Rightarrow x>0$
- Si je suis le recteur de l'UT, alors tu as traversé l'Atlantique à la nage.

Exemple : Evaluation de ¬p ∧ (q ⇒ r)

P	Q	R	¬P	Q ⇒ R	¬p ∧ (q ⇒ r)
V	V	V	F	V	**F**
V	V	F	F	F	**F**
V	F	V	F	V	**F**
V	F	F	F	V	**F**
F	V	V	V	V	**V**
F	V	F	V	F	**F**
F	F	V	V	V	**V**
F	F	F	V	V	**V**

Satisfiabilité et validité

- Soit P une expression booléenne
  - P est satisfaite dans un état si elle a la valeur vrai dans cet état.
  - P est satisfiable s'il y a un état dans lequel elle est satisfaite.
  - P est valide si elle est satisfaite dans tous les états.
  - Une tautologie est une expression booléenne valide.

Exemples

- p ⇒ q est satisfaite dans l'état (p, faux), (q, vrai).
- Par l'item précédent, p ⇒ q est satisfiable.
- p ⇒ q n'est pas valide, car elle n'est pas satisfaite dans l'état (p,vrai), (q,faux).

Modélisation des propositions

- Une proposition est un énoncé qui peut être
  - vrai ou
  - faux
- Exemple :
  - La France a 80 universités et le Sénégal en a 5

Modélisation des propositions

- Pourquoi traduire une proposition en langue naturelle en expression booléenne ?
  - Cela force la résolution des ambiguïtés de la langue;
  - Cela permet de manipuler et de simplifier les expressions selon les lois de la logique propositionnelle;
  - C'est beaucoup plus sûr que de raisonner en langue naturelle.

Traduction triviale en expression booléenne

- Une variable pour toute la proposition. Donnons le nom **P** à la proposition.
- **P** : La France a 80 universités et le Sénégal en a 5
- **P** est une variable booléenne (variable propositionnelle) qui peut prendre la valeur vrai ou faux, selon que la proposition est vraie ou fausse.

Traduction raffinée (sous-propositions)

- Donnons les noms **Q** et **R** aux deux sous-propositions
  - Q : La France a 80 universités
  - R : le Sénégal en a 5
- **R** prise isolément est incensée

Traduction d'une proposition

- Introduire des variables booléennes pour dénoter les sous-propositions.
- Remplacer ces sous-propositions par les variables booléennes correspondantes.
- Traduire le résultat de l'étape 2 en expression booléenne, en utilisant les correspondances «évidentes» entre certains mots et les opérateurs logiques. Cf. table ci- dessous.

Traduction de certains mots en opérateurs

et, mais	deviennent	$\wedge$
ou	devient	$\vee$
ne pas, non	deviennent	$\neg$
il n'est pas vrai que	devient	$\neg$
si p alors q	devient	$p \Rightarrow q$

Traduction de certains mots en opérateurs

- Exemples
  - Si vous ne faites pas les exercices, vous coulerez ;
  - Faites les exercices ou vous coulerez ;
  - Tous les nombres pairs sont divisibles par 4 (peut aussi s'écrire : si un nombre est pair, il est divisible par 4).

Logique propositionnelle

- Calcul
  - méthode de raisonnement au moyen de symboles.
- logique équationnelle ou calcul des propositions
  - Un ensemble d'axiomes (exemple $p \vee q \equiv q \vee p$)
  - Trois règles d'inférence
  - Leibniz : P = Q => E [r := P] = E [r := Q]
  - Transitivité : P = Q, Q = R => P = R
  - Substitution : P, P [r := Q]
- Par conventions
  - P, Q, R, . . . sont des expressions booléennes.
  - p, q, r, . . . sont des variables booléennes.

Théorème et axiome

- Théorème
  - soit un axiome,
  - soit la conclusion d'une règle dont les prémisses sont des théorèmes,
  - soit une expression dont on démontre, en utilisant les règles d'inférence, qu'elle est égale à un axiome ou à un théorème précédemment démontré.
- Axiome
  - expression dont on assume qu'elle est un théorème sans en donner la démonstration. Nous choisirons comme axiomes des expressions valides.

Équivalence et VRAI

- Axiome, associativité de ≡
  - (p≡q)≡r ≡ p≡(q≡r)
- Axiome, commutativité (symétrie) de ≡
  - p≡q≡q≡p
- Identité (élément neutre)
  - Un élément U est l'identité (ou l'élément neutre) d'une opération ◦ ssi b = b◦U = U ◦b, quel que soit b. U est une identité à gauche ssi b = U ◦b pour tout b. U est une identité à droite ssi b = b◦U pour tout b.

Négation, inéquivalence et FAUX

- Axiome, définition de faux : faux ≡ ¬vrai
- Axiome, distributivité de ¬ sur ≡
  - ¬(p≡q) ≡ ¬p≡q
- Axiome, définition de (¬≡)
  - (p (¬≡) q) ≡ ¬(p ≡ q)
- Théorèmes reliant $\equiv, \not\equiv, \neg$ et faux
  - ¬p ≡ q ≡ p ≡ ¬q
  - ¬¬p ≡ p
  - ¬faux ≡ vrai
  - $(p \not\equiv q) \equiv \neg p \equiv q$

Négation, inéquivalence et FAUX

- Théorèmes reliant $\equiv, \not\equiv, \neg$ et faux
  - $\neg p \equiv p \equiv$ faux
  - $(p \not\equiv q) \equiv (q \not\equiv p)$
  - $((p \not\equiv q) \not\equiv r) \equiv (p \not\equiv (q \not\equiv r))$
  - $((p \not\equiv q) \equiv r) \equiv (p \not\equiv (q \equiv r))$
  - $p \not\equiv q \equiv r \equiv p \equiv q \not\equiv r$
- La double négation affirme que la négation est son propre inverse (note : on dit qu'une fonction g est l'inverse d'une fonction f si g(f.x) = x pour tout x).
  - $\neg\neg p \equiv p$

Disjonction

- Elle est définie par 5 axiomes
  - Commutativité (symétrie) de $\vee$
    - $p \vee q \equiv q \vee p$
  - Associativité de $\vee$
    - $(p \vee q) \vee r \equiv p \vee (q \vee r)$
  - Idempotence de $\vee$
    - $p \vee p \equiv p$
  - Distributivité de $\vee$ sur $\equiv$
    - $p \vee (q \equiv r) \equiv p \vee q \equiv p \vee r$
  - Tiers exclu : $p \vee \neg p$ ou encore $p \vee \neg p \equiv$ vrai

Disjonction

- Théorèmessur $\vee$
  - Zéro de $\vee$
    - $p \vee vrai \equiv vrai$
  - Identité de $\vee$
    - $p \vee faux \equiv p$
  - Distributivité de $\vee$ sur $\vee$
    - $p \vee (q \vee r) \equiv (p \vee q) \vee (p \vee r)$
  - $p \vee q \equiv p \vee \neg q \equiv p$

Conjonction

- Axiome ou règle d'or
  - $p \wedge q \equiv p \equiv q \equiv p \vee q$
- Cette règle peut se lire de +sieurs manières
  - $(p \wedge q) \equiv (p \equiv q \equiv p \vee q)$: c'est alors une définition de $\wedge$
  - $(p \equiv q) \equiv (p \wedge q \equiv p \vee q)$: cette lecture indique que p est équivalent à q ssi la conjonction et la disjonction de p et q sont équivalentes.
- Propriétés élémentaires de $\wedge$
  - Commutativité (symétrie) de $\wedge$: $p \wedge q \equiv q \wedge p$
  - Associativité de $\wedge$: $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$

Conjonction

- Propriétés élémentaires de $\wedge$
  - Idempotence de $\wedge$: $p \wedge p \equiv p$
  - Identité de $\wedge$: $p \wedge \text{vrai} \equiv p$
  - Zéro de $\wedge$: $p \wedge \text{faux} \equiv \text{faux}$
  - Distributivité de $\wedge$ sur $\wedge$: $p \wedge (q \wedge r) \equiv (p \wedge q) \wedge (p \wedge r)$
  - Contradiction : $p \wedge \neg p \equiv \text{faux}$
- Théorèmes reliant $\wedge$ et $\vee$
  - Absorption
    - $p \wedge (p \vee q) \equiv p$
    - $p \vee (p \wedge q) \equiv p$
    - $p \wedge (\neg p \vee q) \equiv p \wedge q$
    - $p \vee (\neg p \wedge q) \equiv p \vee q$

Conjonction

- On parle d'absorption parce que la sous-expression q est absorbée par p dans les cas (a) et (b), et parce que ¬p est absorbée et disparaît dans les cas (c) et (d).
- Distributivité de $\vee$ sur $\wedge$: $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
- Distributivité de $\wedge$ sur $\vee$: $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
- De Morgan (lois de De Morgan – Augustus De Morgan)
  - $\neg(p \wedge q) \equiv \neg p \vee \neg q$
  - $\neg(p \vee q) \equiv \neg p \wedge \neg q$

- Théorèmes reliant $\wedge$ et $\equiv$
  - $p \wedge q \equiv p \wedge \neg q \equiv \neg p$
  - $p \wedge (q \equiv r) \equiv p \wedge q \equiv p \wedge r \equiv p$
  - $p \wedge (p \equiv q) \equiv p \wedge q$
  - Remplacement : $(p \equiv q) \wedge (r \equiv p) \equiv (p \equiv q) \wedge (r \equiv q)$

Implication

- Axiome, définition de ⇒ :
  - $p \Rightarrow q \equiv p \vee q \equiv q$
  - alternative de ⇒ : $p \Rightarrow q \equiv \neg p \vee q$
  - alternative de ⇒ : $p \Rightarrow q \equiv p \wedge q \equiv p$
  - Contrapositivité : $p \Rightarrow q \equiv \neg q \Rightarrow \neg p$
- Théorèmes sur ⇒
  - $p \Rightarrow (q \equiv r) \equiv p \wedge q \equiv p \wedge r$
  - Distributivité de ⇒ sur ≡ : $p \Rightarrow (q \equiv r) \equiv p \Rightarrow q \equiv p \Rightarrow r$
  - $p \Rightarrow (q \Rightarrow r) \equiv (p \Rightarrow q) \Rightarrow (p \Rightarrow r)$
  - Transfert : $p \wedge q \Rightarrow r \equiv p \Rightarrow (q \Rightarrow r)$

Implication

- Théorèmes sur ⇒
  - $p \wedge (p \Rightarrow q) \equiv p \wedge q$
  - $p \wedge (q \Rightarrow p) \equiv p$
  - $p \vee (p \Rightarrow q) \equiv vrai$
  - $p \vee (q \Rightarrow p) \equiv q \Rightarrow p$
  - $p \vee q \Rightarrow p \wedge q \equiv p \equiv q$
  - Réflexivité de ⇒ : $p \Rightarrow p \equiv vrai$ ou encore $p \Rightarrow p$
  - Zéro à droite de ⇒ : $p \Rightarrow vrai \equiv vrai$ ou encore $p \Rightarrow vrai$
  - Identité à gauche de ⇒ : $vrai \Rightarrow p \equiv p$
  - $p \Rightarrow faux \equiv \neg p$
  - $faux \Rightarrow p \equiv vrai$ ou encore $faux \Rightarrow p$

Chapitre 2

LOGIQUE DU PREMIER ORDRE

LOGIQUE PROPOSITIONNELLE

Logique des propositions

- Entités
- Prédicats dont les arguments sont des entités
- Connecteurs logiques et, ou, non… $\wedge$, $\vee$, $\neg$

Logique des propositions

- Propositions
  - pleuvoir()
  - prendre(moi, monParapluie)
  - ¬ pleuvoir() ∨ prendre(moi, monParapluie)
  - pleuvoir() ⇒ prendre(moi, monParapluie)

Logique des propositions

- Déduction
- Proposition supposée vraie :
  - ¬(pleuvoir() ∨ faireChaud())
- Proposition déduite :
  - ¬ pleuvoir() ∧ ¬ faireChaud()

- Notation non (pleuvoir() ou faireChaud())
  - ¬ pleuvoir() ∧ ¬ faireChaud()

- Tables de vérité
- Déduction automatique

Logique des propositions

- Déduction
- Propositions supposées vraies :
  - pleuvoir()
  - pleuvoir() ⇒ prendre(moi, monParapluie)
- Proposition déduite :
  - prendre(moi, monParapluie)

- modus ponens
  - pleuvoir()
  - pleuvoir() ⇒ prendre(moi, monParapluie)
  - prendre(moi, monParapluie)

Définition

Extension de la logique propositionnelle qui permet l'usage de variables de types autres que B. Cette extension accroît la puissance de la logique.

Langage naturel

- Tous les convives qui avaient mangé des œufs ont été malades
- Tous les convives qui avaient mangé des œufs n'ont pas été malades
- Aucun convive qui avait mangé des œufs n'a été malade
- Aucun convive qui n'avait pas mangé d'œufs n'a été malade
- Tous les convives qui ont été malades avaient mangé des œufs
- Tous les convives qui ont été malades n'avaient pas mangé d'œufs

- A Wade n'est pas éligible à moins qu'il n'ait démissionné

Formalisme dedans …

- Raisonnement par chaînage avant
- On ajoute la formule déduite à l'ensemble des formules et on recommence
- Raisonnement par chaînage arrière
- On veut savoir si
  - servir(leSamouraï, sushi)
- On suppose que
  - $\forall$ x restaurantJaponais(x) $\Rightarrow$ servir(x, sushi)
- On construit la formule
  - restaurantJaponais(leSamouraï)
- On cherche à savoir si elle est vraie : on recommence

Prédicats et arguments : exemples (1/3)

- êtreNoir(ceChat)
  - êtreNoir() : prédicat
  - ceChat : argument

- dormir(Sidi)
- prendre(Sidi, leParapluieDeSidi) deux arguments
- au-Dessous(température, 0°C)
- frère(Sidi, Mamadou)
- demander(Sidi, Rama, sortirCeSoir) trois arguments
- date(mortDeJean-Paul, 2avril2007)

Prédicats et arguments : exemples (2/3)

- Chaque argument est une variable
- Il peut prendre des valeurs variées
- marquer(Zidane, but) : non (on ne peut pas marquer autre chose)
- marquerBut(Zidane) : oui

- casser(Luc, Marc, figure) : non
- casserLaFigure(Luc, Marc) : oui

- Compositionnalité
- Le sens de la formule est calculable à partir du sens des éléments

Prédicats et arguments : exemples (3/3)

- En général 1 à 3 arguments
- 0 argument :
  - pleuvoir(), faireChaud()
- 4 arguments :
  - parier(Luc, Marie, LucArrivePremier, 15€)
- n arguments :
  - mission(QENO, …, NanterrePréfecture, LaDéfense, CharlesDeGaulleEtoile, Auber, ChâteletLesHalles, GareDeLyon, Nation, Vincennes, ValDeFontenay, NoisyLeGrandMontDEst, NoisyChamps, Noisiel, Lognes, …)

Utilisation

- Confronter une représentation sémantique à une base de faits

- Exemple
- Requête : Est-ce que le vol 123 fait escale à Bombay ?
- Base de faits : la liste des vols avec leurs escales, sous la forme vol(123, Paris, Bombay, Nouméa)
- Réponse : Oui

Représentations non ambiguës

- Les textes sont ambigus
  - Sidi tire sur la poignée
- Les représentations sémantiques ne doivent pas l'être
  - traction(Sidi, laPoignée)
  - tir(Sidi, laPoignée, arme)

Représentations uniques

- Il y a toujours plusieurs façons de dire la même chose
- La représentation sémantique doit être la même
  - Est-ce que le Samouraï fait les sushi ?
  - Est-ce qu'ils ont des sushi au Samouraï ?
  - Est-ce qu'on sert des sushi au Samouraï ?
  - Le Samouraï fait-il des sushi ?

➢ servir(Samouraï, sushi)

Déductions

- Exemple
  - Requête : Est-ce que tous les vols de Paris à Nouméa font escale à Bombay ?
- Base de faits : la liste des vols avec leurs escales
  - vol(234, Paris, Singapour, Nouméa)
- Réponse :
  - Non, le vol 234 de Paris à Nouméa ne fait pas escale à Bombay

Rôles thématiques (1/3)

- Chaque argument a une relation avec le prédicat
- parier(Luc, Marie, LucArrivePremier, 15€)
  - argument 1 parieur 1
  - 2 parieur 2
  - 3 pari du parieur 1
  - 4 enjeu
- Les rôles thématiques sont les étiquettes de ces relations

Rôles thématiques (2/3)

- agent — prendre(Luc, leParapluieDeLuc)
- expérienceur — aimer(Luc, Marie)
- patient — prendre(Luc, leParapluieDeLuc)
- cause — casser(vent, branches)
- résultat — peindre(Luc, paysage, surLeMur)
- contenu — aimer(Luc, sortirAvecMarie)

Rôles thématiques (3/3)

- destination — cacher(Luc, parapluie, dansLePlacard)
- source — sortir(Luc, beurre, frigo)
- position — êtreDans(beurre, frigo)
- instrument — revêtir(Luc, salleDeBains, carrelage)
- bénéficiaire — envoyer(Luc, lettre, Marie)
- détrimentaire — voler(Luc, sac, Marie)

Logique du premier ordre (1/4)

- On ajoute des variables et les quantificateurs

- Entités
- Prédicats dont les arguments sont des entités
- Connecteurs logiques et, ou, non… $\wedge$, $\vee$, $\neg$
- Quantificateurs $\forall$ $\exists$

- Formules
- $\forall$ x restaurant(x) $\wedge$ servir(x, sushi) $\wedge$ êtrePrèsDe(x, laGare)
- $\forall$ x restaurantJaponais(x) $\Rightarrow$ servir(x, sushi)

Logique du premier ordre (2/4)

- Formule --> FormuleAtomique
- | Formule Connecteur Formule
- | non Formule
- | ∀ Variable Formule
- | ∃ Variable Formule
- | (Formule)
- FormuleAtomique --> Predicat (Arguments)
- Arguments --> Argument AutresArguments
- | ε
- AutresArguments --> , Argument AutresArguments
- | ε

Logique du premier ordre (3/4)

- Argument --> Constante
- | Variable
- Connecteur --> et | ou | ⇒
- Constante -->ceChat | Sidi | leParapluie..
- Variable --> x | y | z...
- Predicat -->êtreNoir | dormir...

Logique du premier ordre (4/4)

- ∀ x un "et" sur toutes les valeurs possibles de x
- ∃ x un "ou" sur toutes les valeurs possibles de x

- Déduction par modus ponens
  - ∀ x restaurantJaponais(x) et êtrePrèsDe(x, laGare)
  - ∀ x restaurantJaponais(x) ⇒ servir(x, sushi)
  - ∀ x servir(x, sushi) et êtrePrèsDe(x, laGare)

  - restaurantJaponais(leSamouraï)
  - ∀ x restaurantJaponais(x) ⇒ servir(x, sushi)
  - servir(leSamouraï, sushi)

Formules de la logique des prédicats

- Expressions booléennes dans les quelles certaines variables booléennes sont remplacées par l'un des items suivants :
  - Des prédicats, c'est-à-dire des applications de fonctions booléennes à des arguments de types autres que B . Voici des exemples de prédicats :
    - égale(x, 5), qui écrit en notation infixe : x = 5
    - plus-petit-que(x, y + 5), qui écrit en notation infixe : x < y + 5
  - Des quantifications universelles ou existentielles.
  - Exemple
    - $x < y \wedge x = z \Rightarrow q\,(\,x, z+x\,)$

Axiomatisation de la logique des prédicats

- Axiomes de la logique propositionnelle
- Axiomes des quantifications
- Règles de substitution
- Transitivité
- Leibniz

Quantificateur universel ∀

- La conjonction $\wedge$ est associative, commutative et a vrai comme élément neutre. On peut donc l'utiliser comme quantificateur : $(\wedge x \mid R : P)$
  - Notation : $(\forall x \mid R : P)$
- Prononciations possibles :
  - pour tout x tel que R, P
  - pour tout x satisfaisant R, P
  - tout x satisfaisant R satisfait aussi P
  - quel que soit x, si R alors P.
- Exemple : $(\forall i \mid i>10 : i^2 > 100)$

Quantificateur universel ∀

- Axiome
  - transfert : $(\forall x \mid R : P) \equiv (\forall x \mid : R \Rightarrow P)$
- Théorèmes
  - $(\forall\ x|R:P) \equiv (\forall x|:\neg R \vee P)$
  - $(\forall x|R:P) \equiv (\forall x|:R \wedge P \equiv R)$
  - $(\forall x|R:P) \equiv (\forall x|:R \vee P \equiv P)$
  - $(\forall x|Q \wedge R:P) \equiv (\forall x|Q:R \Rightarrow P$
  - $(\forall x|Q \wedge R:P) \equiv (\forall x|Q:\neg R \vee P)$
  - $(\forall x|Q \wedge R:P) \equiv (\forall x|Q:R \wedge P \equiv R)$
  - $(\forall x|Q \wedge R:P) \equiv (\forall x|Q:R \vee P \equiv P)$

Distributivité et QU ∀ (1)

- Pourvu que ¬libre('x','P')
  - Distributivité de ∨ sur ∀ :
    - $P \vee (\forall x|R:Q) \equiv (\forall x|R:P \vee Q)$
  - Distributivité de ∧ sur ∀ :
    - $\neg(\forall x|:\neg R) \Rightarrow ((\forall x|R:P \wedge Q) \equiv P \wedge (\forall x|R:Q))$
  - $(\forall x \mid R : vrai) \equiv vrai$
  - $(\forall x|R:P \equiv Q) \Rightarrow ((\forall x|R:P) \equiv (\forall x|R:Q))$

Distributivité et QU ∀ (2)

- Manipulation du domaine et du corps
  - $(\forall x|Q \vee R:P) \Rightarrow (\forall x|Q:P)$
  - $(\forall x|R:P \wedge Q) \Rightarrow (\forall x|R:P)$
- Monotonie de ∀
  - $(\forall x|R:Q \Rightarrow P) \Rightarrow ((\forall x|R:Q) \Rightarrow (\forall x|R:P))$

Quantificateur universel ∀

- Elimination du quantificateur universel
  - $(\forall x \mid : P) \Rightarrow P[x := E]$
- Théorèmes et quantification universelle
  - P est un théorème ssi $(\forall x \mid : P)$ est un théorème.

Logiques d'ordre supérieur

- Les arguments d'un prédicat peuvent être des formules avec prédicat et arguments
- date(mort(Jean-Paul), 2avril2007)
- demander(Sidi, Rama, sortir(Sidi, Rama, ceSoir))
- enSilence(quitter(lesEtudiants, laSalle))

Autres notions (1/3)

- Le temps
  - Luc mange des œufs
    1) habituellement
    2) en ce moment
  - Luc mange plusieurs œufs
    1) en même temps
    2) l'un après l'autre
    3) par jour
  - Luc mange déjà des œufs
  - Luc mange encore des oeufs

Autres notions (2/3)

- Modalités
  - Sidi peut manger des œufs
  - Sidi doit manger des œufs
    1) probablement
    2) obligatoirement
  - Sidi mange peut-être des œufs
  - Sidi ne saurait manger des œufs

Autres notions (3/3)

- Coréférence
  - Un des ordinateurs est tombé en panne. Il a été réparé rapidement
  - Tous les ordinateurs ne sont pas restés en état de marche. Il a été réparé rapidement

Qu'êtes-vous supposés savoir et savoir faire ?

Vous devez déjà

- Avoir un discours « juste » sur l'intelligence artificielle
- Etre capable de présenter convenablement le principe et les problèmes de la résolution de problème
- Savoir définir une « heuristique »
- Etre capable d'expliquer le principe du calcul en logique

Qu'êtes-vous supposés savoir et savoir faire ?

Vous devez plus tard

- Etre capable d'écrire un programme de résolution de problème en Prolog
- Savoir définir un système à base de connaissances et décrire un domaine de connaissances simple

PROGRAMMATION LOGIQUE

Histoire en cours d'écriture…

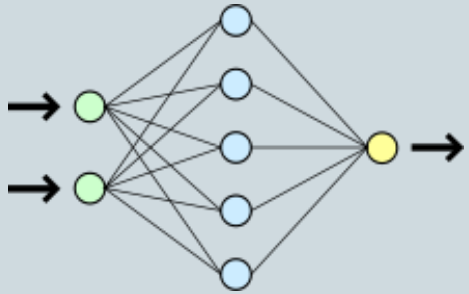
- Acte de naissance : 1956, DarmouthCollege (New Hampshire, USA)
  - John McCarthy (tenant de la logique)
  - Marvin Minsky (tenant d'une approche par schémas)
- Genèse autour de la notion de « machines à penser »
- Comparaison du cerveau avec les premiers ordinateurs

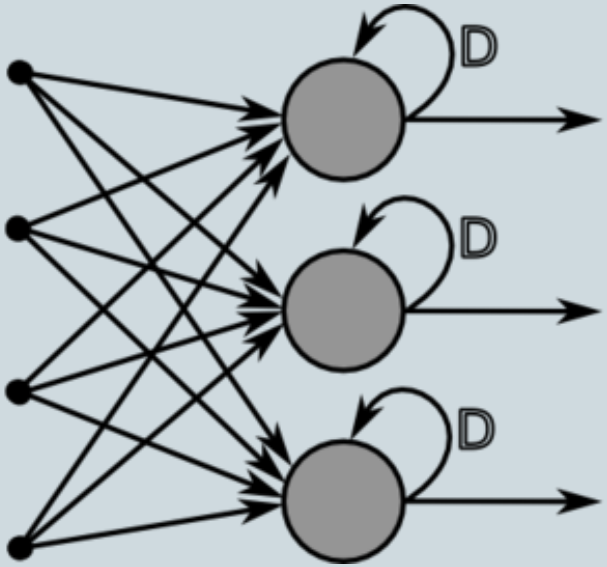
Les grands inspirateurs

- Mc Culloch et Pitts : réseaux neuronaux artificiels (approche physiologique)
- Norbert Wiener : cybernétique
- Claude Shannon : théorie de l'information, entropie
- John Von Neumann : architecture d'un calculateur
- Alan Turing : théorisation des fonctions calculables par machine

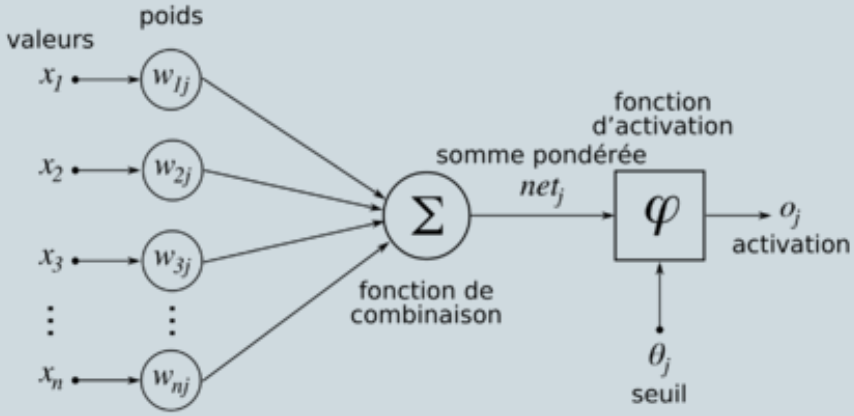
Réseaux de neurones

Vue simplifiée d'un réseau artificiel de neurones





Réseau de neurones
avec rétroaction

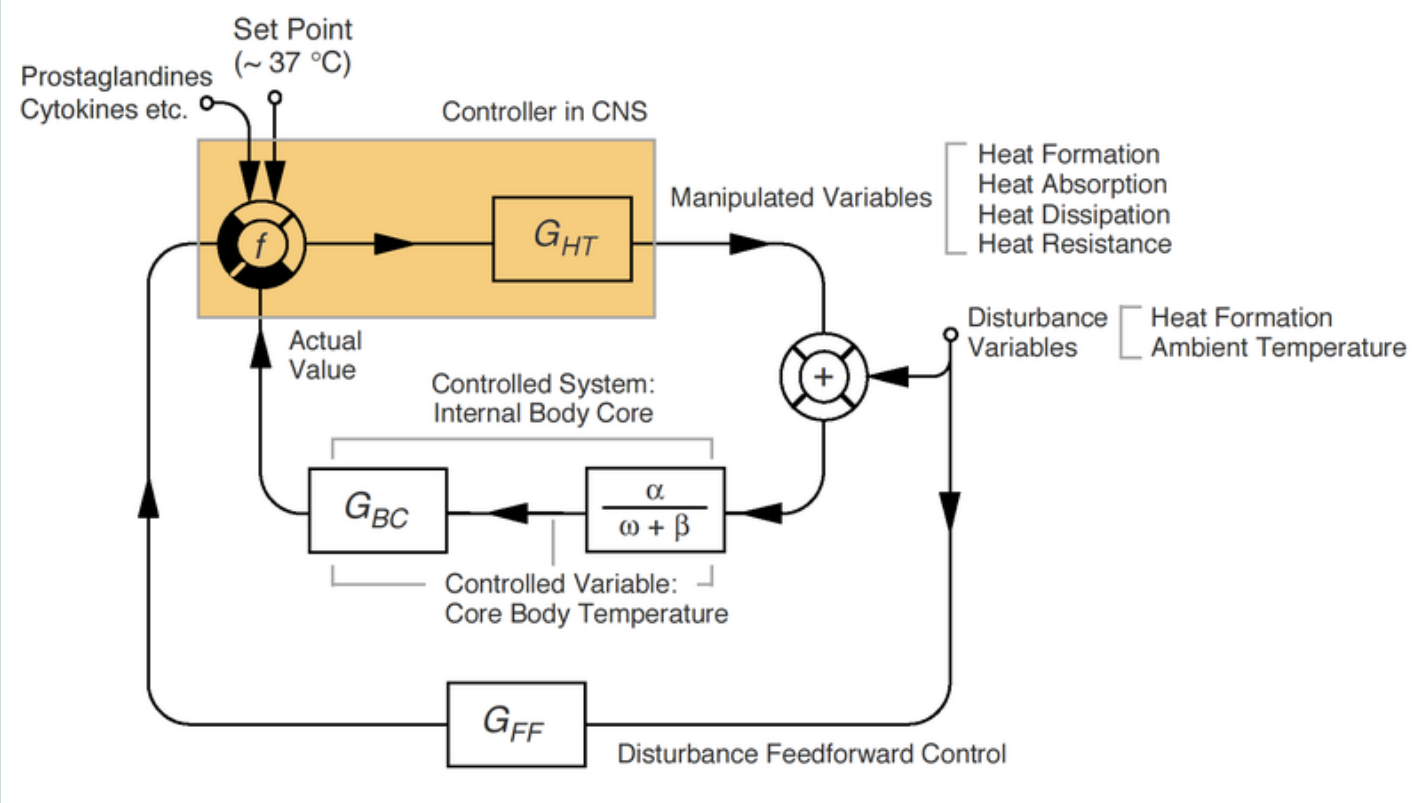


Structure d'un neurone artificiel. Le neurone calcule la somme de ses entrées puis cette valeur passe à travers la fonction d'activation pour produire sa sortie.

Cybernétique

logo

Représentation de la thermorégulation chez les mammifères

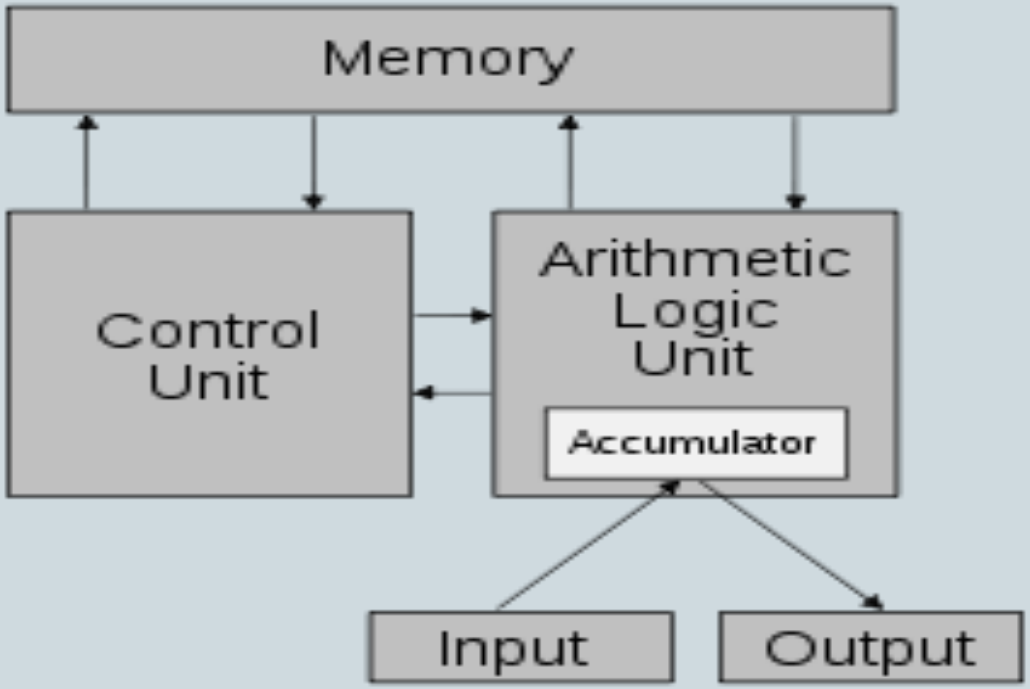


Entropie de Shannon

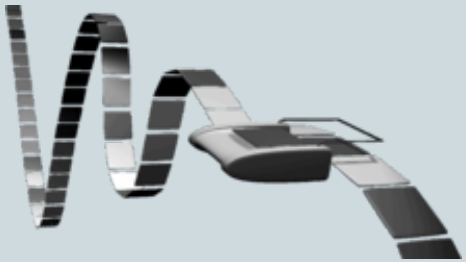
$$H_b(X) = -\mathbf{E}[\log_b P(X = x_i)] = \sum_{i=1}^{n} P_i \log_b \left(\frac{1}{P_i}\right) = -\sum_{i=1}^{n} P_i \log_b P_i.$$

$$\begin{aligned}
H(X,Y) &= -\sum_{x\in X}\sum_{y\in Y} P(x,y)\log P(x,y) \\
&= -\sum_{x\in X}\sum_{y\in Y} P(x)P(y)\log\left[P(x)P(y)\right] \\
&= -\sum_{x\in X}\sum_{y\in Y} P(x)P(y)\left[\log P(x) + \log P(y)\right] \\
&= -\sum_{x\in X}\sum_{y\in Y} P(x)P(y)\log P(x) - \sum_{y\in Y}\sum_{x\in X} P(x)P(y)\log P(y) \\
&= -\sum_{x\in X} P(x)\log P(x)\sum_{y\in Y} P(y) - \sum_{y\in Y} P(y)\log P(y)\sum_{x\in X} P(x) \\
&= -\sum_{x\in X} P(x)\log P(x) - \sum_{y\in Y} P(y)\log P(y) \\
&= H(X) + H(Y)
\end{aligned}$$

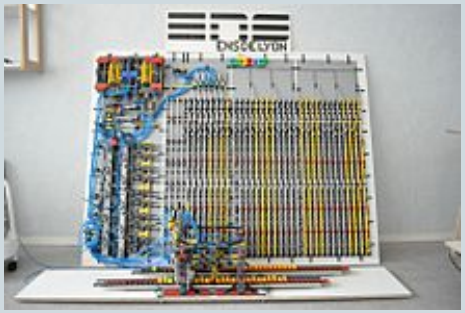
Modèle de Neumann



Machine de Turing



Vue d'artiste d'une Machine de Turing (sans la table de transition)



Machine créée lors de l'année Turing

Les premiers programmes d'IA

- Newell, Simon et Shaw proposent un premier programme de démonstration de théorèmes en logique (1956!)

- Ils généralisent en proposant le General Problem Solver qui progresse dans la résolution en évaluant la différence entre la situation du solveur et le but à atteindre.

Premiers défis…

- Programmes capables de jouer aux échecs (premières idées en 1950 par Shannon!) -> première victoire sur un maître en 1997 Deep Blue bat Kasparov (wikipedia)
- Test « d'intelligence » (Evans 1963) : trouver la suite d'une série de figures.
- Résolution de problèmes par propagation de contraintes (Waltz 1975)
- Dialogue en « langage naturel » (Eliza, Weizenbaum 1965) (Système SHRDLU, Winograd 1971)

L'ère des « systèmes experts »

- Les années 70 et 80 vivent un véritable engouement pour les systèmes experts:
  - DENDRAL (en chimie)
  - MYCIN (en médecine)
  - Hersay II (en compréhension de la parole)
  - Prospector (en géologie)
- Générateurs de systèmes experts
  - GURU
  - CLIPS

Langages de programmation pour l'IA ?

- LISP (origine américaine)
- PROLOG (France ! Colmerauer)
- SmallTalk (Langage objet)
- Les langages de Frame
  - YAFOOL (YetAnother Frame based Object Oriented Language)
  - KL-ONE (Knowledge Language)
- Langage de logique de description

Le projet de Doug Lenat

- Doug Lenat imagine un système capable d'apprendre continuement… toutes les connaissances que l'on peut lui fournir !

- Le serveur de connaissances CYC = une encyclopédie « intelligente » (à visiter!)
  - Ce n'est pas le rêve de Doug Lenat qui imaginait vraiment stocker les connaissances.
  - C'est une encyclopédie anglosaxonne avec pas mal de possibilités -> produit semi-commercial

Nouvelles questions de l'IA

- L'informatique c'est maintenant le WEB ! L'IA l'habite déjà et en façonne le futur → WWW conférences
  - S'adapter à des situations dynamiques, changeantes, singulières…
  - Assister l'apprentissage humain !
  - Gérer des dialogues entre « agents » hétérogènes
  - Voir la cognition comme une émergence dans l'interaction avec l'environnement
    > Concevoir une nouvelle génération de systèmes informatiques
    > Imaginer des systèmes qui sont conçus dans la continuité sur la base des usages
    > Cognition située, distribuée, émergente …

Multiples facettes de l'IA

- Facette des mathématiques
  - formalisation du raisonnement mathématique (logique)
  - Contribution à de nouveaux champs (logiques modales -> logique possibiliste)
- Facette informatique
  - Nouveau « paradigmes » de programmation
    - Programmation logique
    - Programmation objet
    - Programmation fonctionnelle

Multiples facettes de l'IA

- Facette informatique
  - Nouvelles façons de voir les systèmes d'information
    - Gestion de la connaissance
    - Indexation WEB (semantic web)
    - Description des documents numériques
  - Applications nombreuses
    - Aide à la décision
    - Aide au diagnostic
    - Aide à la planification
    - Aide au traitement automatique de la langue
    - Aide à la conception
    - ...
  - Omniprésence dans les Systèmes de Traitement de l'Information et de la Communication (STIC)

Avons vu – Aurons à voir

1. Résolution automatique de problème
   - Recherche de solution dans un espace d'états
2. Méthodes de calcul en logique
   - Où comment on peut construire un raisonnement par reformulations successives
   - Les questions de complétude, de formalisation et d'applicabilité
3. Un langage de programmation logique
   - Principe
   - Syntaxe
   - Sémantique
   - Mise en œuvre

   > programmation des autres aspects vus en cours

UN LANGAGE DE PROGRAMMATION *LOGIQUE*

Le langage Prolog
I - les bases

Plan

1. Historique
2. Données, relations et faits
3. Prédicats et formules
4. Règles
5. Clauses de Horn
6. Démonstration en Prolog
7. Stratégie de Prolog

Historique

- 1972 : création de Prolog par A. Colmerauer et P. Roussel à Luminy.
- 1980 : reconnaissance de Prolog comme langage de développement en Intelligence Artificielle
- Depuis plusieurs versions, dont une tournée vers la programmation par contraintes.

Les données

- Constantes
  - Symbole : chaîne de caractères (minuscule)
  - Nombres : entiers ou flottants
- Variables : chaîne de caractères (majuscule)
  - Exprimer une propriété concernant une catégorie d ’objets
  - Interroger le système Prolog à l ’aide d ’une question

Les relations (1)

- Propriété qui lie un certain nombre d ’objets
  - *la possession lie le propriétaire et l ’objet possédé*
- Propriétés des relations :
  - caractère ordonné
  - transitive : si a R b et b R c alors a R c
  - symétrique : si a R b alors b R a
  - nom commençant par une minuscule

Les relations (2)

- Utilité des relations :
  - lien entre objets
  - propriété d ’un objet
- Plusieurs possibilités pour établir la même relation
  - *Hélène est une fille de cinq ans* peut s ’exprimer de deux façons.

Les faits

- Affirmation de l ’existence d ’une relation entre certains objets
  - *La chèvre est un animal herbivore*
  - *le loup est un animal cruel*
- Formule vraie à priori
- Constitue une partie des données d ’un problème

Prédicats et formules(1)

- En Prolog, une relation possède :
  - un nom
  - un nombre d ’arguments
    - *couleur(voiture, rouge)*
- En logique, relation = **prédicat**
  - Couleur
- Application du prédicat à ses arguments = **formule**

Prédicats et formules (2)

- Nombre d ’arguments du prédicat = **arité**
  - unaire → propriété de l ’argument
    - Garcon(jerome)
  - arité zéro → signification logique très restreinte
    - p()
- Exercice : écrire les faits présentés précédemment sous forme de prédicats
    - chèvre, loup

Règles Prolog (1)

- Enoncent la dépendance d'une relation entre objets par rapport à d'autres relations
- Concernent des catégories d 'objets / faits
  - fait : *fille(helene)*
  - règles : *si X est une fille alors X aime les poupées* qui s'écrit : ***aime(X, poupees) :- fille(X)***.
- Le « **si** » s'écrit « **:-** » en Prolog

Règles Prolog (2)

- Il peut y avoir plusieurs conditions derrière le « :- », séparées par des virgules
  - Exemple : *un animal qui vole est un oiseau*
  - ***oiseau(X) :- animal(X), voler(X)***.
- Exercice :
  - *la chèvre est un animal herbivore*
  - *le loup est un animal cruel*
  - *un animal cruel est carnivore*
  - *un animal carnivore mange de la viande*

Règles Prolog (3)

- Suite de l'exercice :
  - *un animal herbivore mange de l'herbe*
  - *un animal carnivore mange des animaux herbivores*
  - *les carnivores et les herbivores boivent de l'eau*
  - *un animal consomme ce qu'il boit ou ce qu'il mange*

  - Question : y a-t-il un animal cruel et que consomme-t-il ?
  - Idée : identifier les objets, les faits, les règles

Exercices d'applications

- animal(chevre).
- animal(loup).
- herbivore(chevre).
- cruel(loup).
- carnivore(X):-cruel(X).
- manger(X, viande):-animal(X), carnivore(X).
- manger(X, herbe):-animal(X), herbivore(X).
- manger(X, Y):-animal(X), animal(Y), herbivore(Y), carnivore(X).
- boire(X, eau):-animal(X), carnivore(X).
- boire(X, eau):-animal(X), herbivore(X).
- consomme(X, A):-boire(X,A).
- consomme(X, A):-manger(X,A).

CLAUSES DE HORN

Pr. THIAM, Maître de conférences informatique

Clauses de Horn (1)

- Ce sont les faits et les règles.
- Forme générale : F :- F1, F2, …, Fn.
  - Se traduit par « F si F1 et F2 et…et Fn »
  - F : formule atomique
  - Fi : littéraux (formules atomiques ou leur négation)
- En Prolog : pour démontrer F, il faut démontrer F1, F2, …, et Fn.

Clauses de Horn (2)

- F est la **tête** de la clause
- F1, F2, …, Fn est appelée la **queue** de la clause
- Un fait est une clause dont la queue est vide
  - Exemples :
    - *naissance(prolog, 1972).*
    - *aime(X, bach).* : « Pour tout X, X aime Bach. »

Clauses de Horn (3)

- Une règle est une clause dont la queue est non vide. La plupart des règles contiennent des variables.
- Définition d'une variable anonyme : « _ »
  - *a_un_salaire(X) :- employeur(Y,X).* peut s'écrire : *a_un_salaire(X) :- employeur(_,X).*
- Déclaration d'un prédicat = ensemble de faits et de règles

PROGRAMME & DEMONSTRATION

Programme

- Programmes Prolog : succession de déclarations de prédicats
- Pas d'ordre à respecter
- Possibilité de plusieurs fichiers
  - Exemple de la famille : prédicats *homme*, *femme* et *enfant*
    - *enfant(X,Y,Z)*
    - *enfant(X,Y) :- enfant(X,Y,_).* ou *enfant(X,_,Y).*

Démonstration

- A partir d ’un programme, on peut poser des questions
  - Ex : *frere(patrick, X)*. Pour trouver les frères de P.
  - C’est-à-dire déterminer les valeurs des variables (Ex : *X*) telles que la question soit déductible des faits et prédicats du programme.
- On parle de **résolution de question** ou de **démonstration de question**.

Unification (1)

- Exemple :
  - *frere(X,Y) :- homme(X), enfant(X,Z), enfant(Y,Z), X\=Y.*

    où \= représente le prédicat de différence.
  - *frere(Patrick, Qui)* : tentative d'unification avec la tête de la clause *frere(X,Y)*

- **Définition** : procédé par lequel on essaie de rendre deux formules identiques en donnant des valeurs aux variables qu'elles contiennent.

Unification (2)

- Résultat : c'est un **unificateur** (ou **substitution**), un ensemble d'affectations de variables.
  - Exemple : *{X=Patrick, Qui=Y}*
- Le résultat n'est pas forcément unique, mais représente l'unificateur le plus général.
- L'unification peut réussir ou échouer.
  - *e(X,X)* et *e(2,3)* ne peuvent être unifiés.

Unification (3)

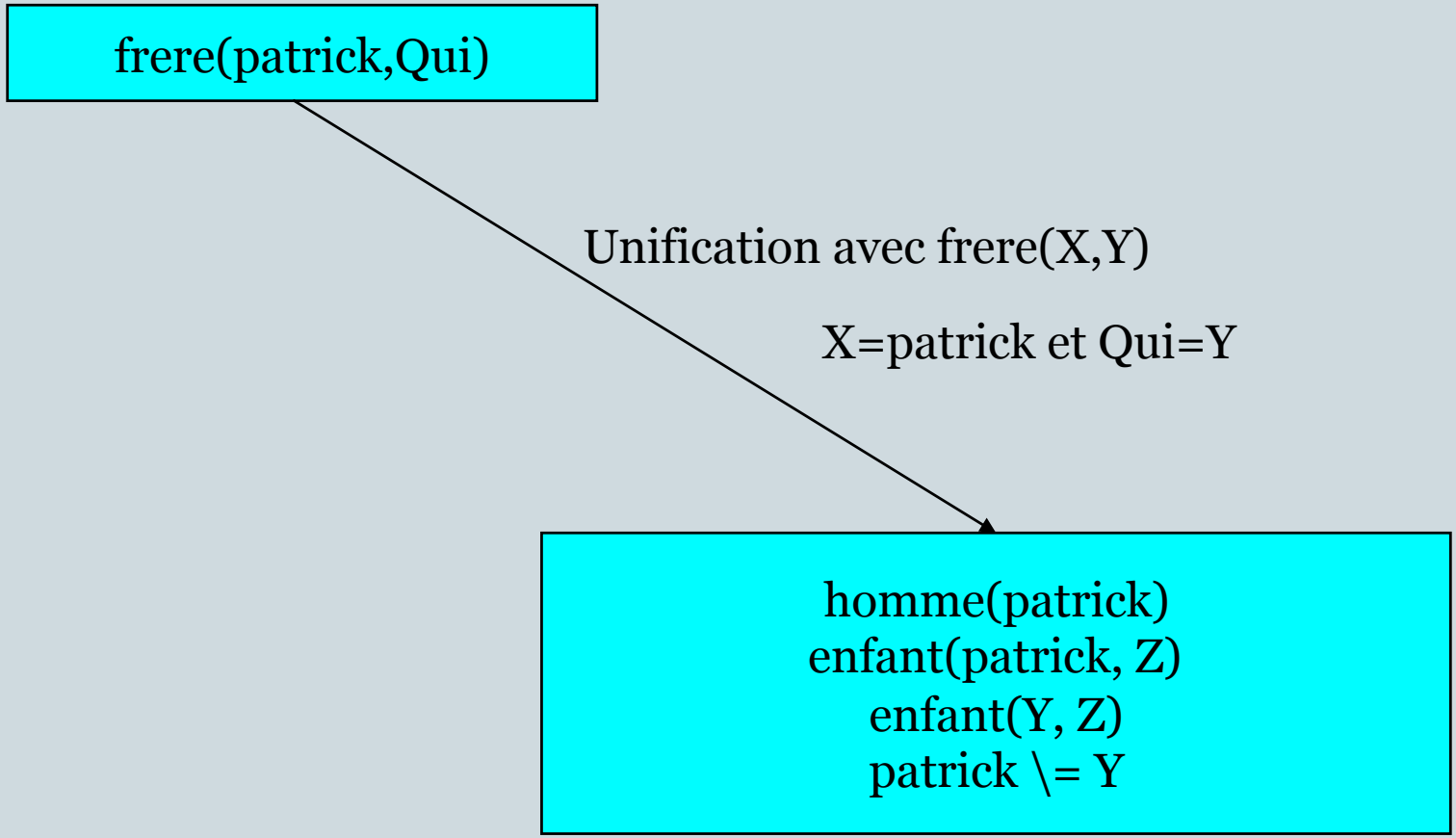
- Prédicat d'unification : « = »
  - *a(B,C) = a(2,3).* donne pour résultat :
    YES {B=2, C=3}
  - *a(X,Y,L) = a(Y,2,carole).* donne pour résultat:
    YES {X=2, Y=2, L=carole}
  - *a(X,X,Y) = a(Y,u,v).* donne pour résultat :
    NO

Pas de démonstration (1)

- Si l ’unification échoue : situation d'échec sur la règle considérée pour démontrer la formule.
- Si l’unification réussit : substitution des variables présentes dans la queue de la clause par les valeurs correspondantes des variables de l ’unificateur.

Pas de démonstration (2)

frere(X,Y) :- homme(X), enfant(X,Z), enfant(Y,Z), X\=Y.



Pas de démonstration (3)

- Démonstration de cet ensemble de formules dans l'ordre
- Enrichissement du système avec les valeurs obtenues des variables.

1. homme(patrick)
2. enfant(patrick, Z)
3. enfant(Y,Z)
4. patrick\=Y

- A la fin, l'ensemble des couples valeur-variable des variables présentes dans la question initiale forme la ***solution*** affichée par Prolog.

Points de choix

- Plusieurs règles concernant une même question :
  - essais consécutifs dans l'ordre de déclaration
  - représentation sous forme d'arbre
  - nœuds de l'arbre appelés **points de choix**

- Exemple : arbre binaire pour répondre à la question
  - *enfant(sidi, samba)*

Arbre de recherche (1)

- On parle d'arbre de recherche d'une question
  - Racine de l'arbre : **question**
  - Nœud : **points de choix** (formule à démontrer)
  - Passage d'un nœud vers son fils en considérant l'une des règles et en effectuant
    - une unification et
    - un pas de démonstration

Arbre de recherche (2)

- Nœuds de gauche à droite dans l'ordre de déclaration des règles
- **Nœuds d'échec** : aucune règle ne permet de démontrer la 1ère formule du nœud
- **Nœuds de succès** : ne contient plus aucune formule, tout a été démontré et les éléments de solution sont trouvés en remontant vers la racine de l'arbre

Stratégie de Prolog (1)

- Pour résoudre une question, Prolog construit l'arbre de recherche de la question

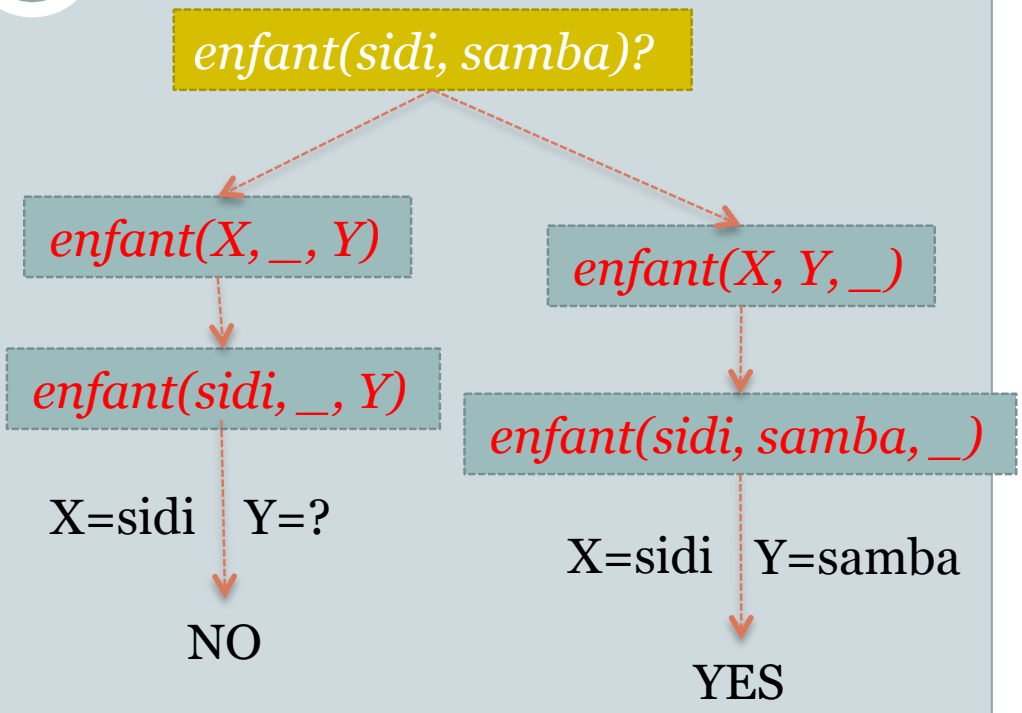
- Parcours en profondeur d'abord
  - **nœud de succès** : c'est une solution, Prolog l'affiche et cherche d'autres solutions
  - **nœud d'échec** : remontée dans l'arbre jusqu'à un point de choix possédant des branches non explorées

Stratégie de Prolog (2)

- On parle de **backtrack**. Si un tel nœud de choix n'existe pas, la démonstration est terminée, il n'y a pas d'autre solution.
- Possibilité de branche infinie et donc de recherche sans terminaison… Attention à :
  - ordre des littéraux dans la queue de clause
  - ordre des clauses

Données

- *homme(samba).*
- *homme(sidi).*
- enfant(sidi, samba, _).
- enfant(X,Y):-enfant(X,_,Y).
- enfant(X,Y):-enfant(X,Y,_).



UN LANGAGE DE PROGRAMMATION *LOGIQUE*

Le langage Prolog
II - aspects avancés

Plan

1. Structures de données
2. Opérateurs et expressions arithmétiques
3. Evaluation des expressions et prédicats de comparaison
4. La coupure et la négation
5. Entrées-sorties

Plan

1. Structures de données
2. Opérateurs et expressions arithmétiques
3. Evaluation des expressions et prédicats de comparaison
4. La coupure et la négation
5. Entrées-sorties

Structures de données

- Déjà vu : objets atomiques (nombres et variables)
- Il existe aussi une structure de données plus complexe appelée arbre Prolog ou structure Prolog.
- Prolog connaît aussi les listes. Ces structures feront l'objet d'un chapitre de ce cours.

Plan

1. Structures de données
2. Opérateurs et expressions arithmétiques
3. Evaluation des expressions et prédicats de comparaison
4. La coupure et la négation
5. Entrées-sorties

Les opérateurs (1)

- Ils permettent d'utiliser une syntaxe infixée, préfixée ou suffixée pour écrire des prédicats.
- Ce terme n'est pas réservé aux opérateurs arithmétiques (+, - , * et / sont des opérateurs infixés).
- Ils ont des priorités différentes.

Les opérateurs (2)

- Trois types d'opérateur :
  - binaire infixé : il figure entre ses 2 arguments et désigne un arbre binaire

    X + Y
  - unaire préfixé : il figure avant son argument et désigne un arbre unaire

    -X
  - unaire suffixé : il figure après son argument et désigne un arbre unaire (Ex : X^2)

Les opérateurs (3)

- Associativité :
  - A gauche :
    - **X op Y op Z** est lu comme **(X op Y) op Z**
  - A droite :
    - **X op Y op Z** est lu comme **X op (Y op Z)**
  - Non associatif : les parenthèses sont obligatoires
    - la syntaxe **X op Y op Z** est interdite

Les opérateurs (4)

- Déclaration des opérateurs :
  - possibilité de modifier la syntaxe Prolog en définissant de nouveaux opérateurs
  - définition : par l'enregistrement de faits de la forme suivante ***op(Priorite, Specif, Nom)***
    - Nom : nom de l'opérateur
    - Priorite :compris entre 0 (le plus prioritaire) et 1200
    - Specif :type de l'opérateur (infixé, associatif…)

Les opérateurs (5)

- Exemple de déclaration d'un opérateur :
  - op(1000, xfx, aime) définit un opérateur infixé non associatif aime
  - Prolog traduira une expression du type X aime Y en le terme aime(X, Y), et si
  - Prolog doit afficher le terme aime(X, Y), il affichera X aime Y.
  - Exercice: définissez les opérateurs et (conjonction), ou (disjonction), non (négation) et => (implication) afin de pouvoir écrire par ex : (A ou non A)…

Les expressions arithmétiques (1)

- Prolog connaît les entiers et les nombres flottants.
- Expressions arithmétiques construites à partir de nombres, de variables et d'opérateurs arithmétiques.
- Evaluation : par l'utilisation de l'opérateur ***is*** par exemple dans **X is 3 - 2**.

Les expressions arithmétiques (2)

- Opérations habituelles : addition, soustraction, multiplication, division entière (symbole //), division flottante (symbole /).
- Selon les systèmes Prolog, différentes fonctions mathématiques comme abs(X), ln(X), sqrt(X)
- Représentées par des arbres Prolog

Les expressions arithmétiques (3)

- Expressions et unification : attention à certaines tentatives d ’unification
  - la tentative d’unification entre 3+2 et 5 échouera. En effet, l’expression 3+2 est un arbre alors que 5 est un nombre.
  - L'évaluation des expressions ne fait pas partie de l’algorithme d’unification.

Plan

1. Structures de données
2. Opérateurs et expressions arithmétiques
3. Evaluation des expressions et prédicats de comparaison
4. La coupure et la négation
5. Entrées-sorties

Les prédicats de comparaison

- Comparaison des expressions arithmétiques
  - X =:= Y se traduit par X est égal à Y
  - X \= Y se traduit par X est différent de Y
  - X < Y
  - X =< Y
  - X > Y
  - X >= Y
- Il y a évaluation puis comparaison.

Plan

1. Structures de données
2. Opérateurs et expressions arithmétiques
3. Evaluation des expressions et prédicats de comparaison
4. La coupure et la négation
5. Entrées-sorties

Notion de coupure (1)

- Différents noms : coupure, **cut** ou **coupe-choix**
- Introduit un contrôle du programmeur sur l'exécution de ses programmes
  - en élaguant les branches de l'arbre de recherche
  - rend les programmes plus simples et efficaces
- Différentes notations : ! ou / sont les plus courantes

Notion de coupure (2)

- Le coupe-choix permet de signifier à Prolog qu'on ne désire pas conserver les points de choix en attente
- Utile lors de la présence de clauses exclusives
  - Ex : les 2 règles suivantes
    - humain(X) :- homme(X). s'écrit humain(X) :- homme(X)!.
    - humain(X) :- femme(X). s'écrit humain(X) :- femme(X)!.

Notion de coupure (3)

- Le coupe-choix permet :
  - d'éliminer des points de choix
  - d'éliminer des tests conditionnels que l'on sait inutile

  => plus d'efficacité lors de l'exécution du programme

- Quand Prolog démontre un coupe-choix,
  - il détruit tous les points de choix créés depuis le début de l'exploitation du paquet des clauses du prédicat où le coupe-choix figure.

Notion de coupure (4)

- Exemples d'utilisation :
  - Pour prouver que a est un élément d'une liste, on peut s'arrêter dès que la première occurrence de a a été trouvée.
  - Pour calculer la racine carrée entière d'un nombre, on utilise un générateur de nombres entiers entier(k). L'utilisation du coupe-choix est alors indispensable après le test K*K>N car la génération des entiers n'a pas de fin.
- entier(0).
- entier(N) :- entier(N1), N is N1+1.
- racine(N, R) :- entier(K), K*K >N, ! , R is K-1.

Notion de coupure (5)

- Repeat et fail :
  - Le prédicat fail/0 est un prédicat qui n'est jamais démontrable, il provoque donc un échec de la démonstration où il figure.
  - Le prédicat repeat/0 est une prédicat prédéfini qui est toujours démontrable mais laisse systématiquement un point de choix derrière lui. Il a une infinité de solutions.
  - L'utilisation conjointe de repeat/0, fail/0 et du coupe-choix permet de réaliser des boucles.

Notion de coupure (6)

- Dangers du coupe-choix :
  - Considérez les programmes suivants :
    - enfants(helene, 3). → enfants(helene, 3) :- !.
    - enfants(corinne, 1). → enfants(corinne,1) :- !.
    - enfants(X, 0). → enfants(X, 0).
  - Interrogation enfant(helene, N). dans les deux cas.
    - N=3 et N=0
    - N=3 mais enfant(helene, 3). donne YES.
- Le programme correct serait :
  - enfant(helene,N) :- !, N=3.

Notion de coupure (fin)

- En résumé, les utilités du coupe-choix sont :
  - éliminer les points de choix menant à des échecs certains
  - supprimer certains tests d'exclusion mutuelle dans les clauses
  - permettre de n'obtenir que la première solution de la démonstration
  - assurer la terminaison de certains programmes
  - contrôler et diriger la démonstration

La négation (1)

- Problème de la négation et de la différence en Prolog
- Pas de moyen en Prolog de démontrer qu'une formule n'est pas déductible.

La négation (2)

- Négation par l'échec
  - Si F est une formule, sa négation est notée not(F) ou not F. L'opérateur not est préfixé associatif.
  - Prolog pratique la négation par l'échec, c'est-à-dire que pour démontrer not(F) Prolog va tenter de démontrer F. Si la démonstration de F échoue, Prolog considère que not(F) est démontrée.
  - Pour Prolog, not(F) signifie que la formule F n'est pas démontrable, et non que c'est une formule fausse. C 'est ce que l'on appelle l'hypothèse du monde clos.

La négation (3)

- Elle est utilisée pour vérifier qu'une formule n'est pas vraie.
- La négation par l'échec ne doit être utilisée que sur des prédicats dont les arguments sont déterminés et à des fins de vérification
  - Son utilisation ne détermine jamais la valeur d'une variable

La négation (4)

- Dangers :
  - Considérez le programme suivant :
- p(a).
  - Et interrogez Prolog avec :
  - ?- X = b, not p(X).
  - YES {X = b}
- Prolog répond positivement car p(b) n'est pas démontrable.

La négation (5)

- Par contre, interrogez le avec :
  - ?- not p(X), X=b.
  - NO
- Prolog répond négativement car p(X) étant démontrable avec {X = a}, not p(X) est considéré comme faux.

  => Incohérence qui peut être corrigée si l'on applique la règle vue précédemment : n'utiliser la négation que sur des prédicats dont les arguments sont déterminés.

La négation (6)

- Le coupe-choix et la négation :
  - not P :- call(P), !, fail.
  - not P.

- Le prédicat prédéfini call/1 considère son argument comme un prédicat et en fait la démonstration. Pour démontrer not P, Prolog essaie de démontrer P à l 'aide de call(P). S'il réussit, un coupe-choix élimine les points de choix éventuellement créés durant cette démonstration puis échoue. Le coupe-choix ayant éliminé la deuxième règle, c'est la démonstration de not P qui échoue. Si la démonstration de call(P) échoue, Prolog utilise la deuxième règle qui réussit.

La négation (7)

- L'unification :
  - Prédicat binaire infixé : X = Y
  - Pour démontrer X = Y, Prolog unifie X et Y ; s'il réussit, la démonstration est réussie, et le calcul continue avec les valeurs des variables déterminées par l'unification.
- Exemple
  - ?- X = 2.
  - YES
  - ?- X = 2, Y = 3, X = Y.
  - NO

La négation (fin)

- La différence est définie comme le contraire de l'unification
  - Elle est notée : X \= Y. Elle signifie que X et Y ne sont pas unifiables, et non qu'ils sont différents.
    - Ex : Z \= 3. Sera faux car Z et 3 sont unifiables.
  - Elle peut être définie comme:
    - X \= Y :- not X = Y.
  - ou
    - X \= X :- !, fail.
    - X \= Y.

Plan

1. Structures de données
2. Opérateurs et expressions arithmétiques
3. Evaluation des expressions et prédicats de comparaison
4. La coupure et la négation
5. Entrées-sorties

Les entrées-sorties (1)

- Ecriture sur l'écran ou dans un fichier
- Lecture à partir du clavier ou d'un fichier
- Affichage de termes :
  - write(1+2) affiche 1+2
  - write(X). affiche X sur l'écran, sous la forme _245 qui est le nom interne de la variable.
  - nl/0 permet de passer à la ligne
  - tab/1 tel que tab(N) affiche N espaces

Les entrées-sorties (2)

- Affichage de termes (suite) :
  - display/1 agit comme write/1 mais en affichant la représentation sous forme d'arbre
  - Ex :
    - write(3+4), nl, display(3+4), nl.
  - Affiche :
    - 3+4
    - +(3,4)
    - YES

Les entrées-sorties (3)

- Affichage de caractères et de chaînes
  - put/1 s'utilise en donnant le code ASCII d'un caractère :
  - put(97).
  - affiche : a
- Prolog connaît aussi les chaînes de caractères. Elles sont notées entre " ". Mais pour Prolog la chaîne est une liste de codes ASCII des caractères la composant.

Les entrées-sorties (4)

- Lecture de termes :
  - read/1 admet n'importe quel terme en argument.
    - Il lit un terme au clavier et l'unifie avec son argument. Le terme lu doit être obligatoirement suivi d'un point. Certains systèmes Prolog affichent un signe d'invite lorsque le prédicat read/1 est utilisé.
- Exemple :
  - ?- read(X).
  - : a(1,2).
  - YES {X = a(1,2)}

Les entrées-sorties (5)

- Autre exemple :
  - calculateur :- repeat, (*boucle*)
  - read(X), (*lecture expression*)
  - eval(X,Y), (*évaluation*)
  - write(Y), nl, (*affichage*)
  - Y = fin, !. (*condition d'arrêt*)
  - eval(fin, fin) :- !. (*cas particulier*)
  - eval(X, Y) :- Y is X. (*calcul d'expressions*)

- lit des expressions arithmétiques, les évalue et les imprime jusqu'à ce que l'utilisateur rentre "fin" au clavier.

Les entrées-sorties (6)

- Le prédicat eval/2 traite le cas particulier de l'atome fin. Lorsque fin est lu, le coupe-choix final est exécuté et met fin à la boucle. Sinon Y = fin échoue et le prédicat retourne en arrière jusqu'à repeat/0.

- Exemple d'utilisation :
  - ?- calculateur.
  - : 2+3 . (*noter l'espace entre 3 et . Pour éviter de penser qu'il sagit d'un réel*)
  - 5
  - : fin.
  - fin
  - YES

Les entrées-sorties (7)

- Lecture de caractères :
  - get/1 et get0/1. Tous les deux prennent en argument un terme unifié avec le code ASCII du caractère lu. Si l'argument est une variable, celle-ci prendra pour valeur le code ASCII du caractère lu. get/1 ne lit que les caractères de code compris entre 33 et 126.
- Les fichiers :
  - Un fichier peut être ouvert en lecture ou écriture.

Les entrées-sorties (8)

- En écriture :
  - mode write : son contenu est effacé avant que Prolog y écrive.
  - mode append : Prolog écrira à partir de la fin du fichier.
- Ouverture d'un fichier : prédicat open/3
  - argument 1 : nom du fichier
  - argument 2 : mode d'ouverture write, append ou read
  - argument 3 : variable qui va recevoir un identificateur de fichier appelé flux ou stream. (dépend du système Prolog utilisé).

Les entrées-sorties (9)

- Tous les prédicats read, write et autres vus auparavant admettent un second argument : le flux identifiant le fichier.
  - Ex : write(Flux, X). où Flux est un identificateur de fichier
  - Ex : read(Flux,X), get(Flux, X), get0(Flux,X)
  - Fermeture du fichier : prédicat close/1 qui prend en argument le flux associé au fichier.

Les entrées-sorties (fin)

- Exemple d'utilisation
  - ecrire(T) :-
  - open(' a.prl ', append, Flux), (*ouverture*)
  - write(Flux, T), nl(Flux), (*écriture*)
  - close(Flux). (*fermeture*)

Plan

1. Structures de données
2. Opérateurs et expressions arithmétiques
3. Evaluation des expressions et prédicats de comparaison
4. La coupure et la négation
5. Les prédicats retardés (facultatif)
6. Entrées-sorties

Les prédicats retardés

- Prédicats dif/2 et freeze/2
  - Introduits dans le système Prolog II par A. Colmerauer et M. Van Caneghem
  - Apportent une plus grande souplesse dans l'utilisation des prédicats arithmétiques
  - Préservent la modularité des algorithmes classiques de Prolog tout en permettant leurs optimisations

La différence retardée

- Comparaison entre dif(X, Y) et X\=Y
  - Pas de différence si X et Y sont instanciées
  - Si X ou Y n ’est pas instanciée, le prédicat dif/2 autorise Prolog à retarder la vérification de la différence jusqu’à ce que les deux variables soient instanciées.
  - Permet de poser des contraintes de différence sur les variables. On parle de contraintes passives car elles sont simplement mémorisées.

Le gel des prédicats

- Généralisation de la différence retardée à n'importe quel prédicat.
- Syntaxe : freeze(Variable, Formule) signifie que cette formule sera démontrée lorsque cette variable sera instanciée.
  - Par exemple :
  - dif(X, Y) :- freeze(X, freeze(Y, X\=Y))

Les prédicats retardés (fin)

- Danger :
  - c'est à vous de vous assurer que toutes les formules seront effectivement démontrées. Certains Prolog signalent les prédicats qui n'ont pas été dégelés suite à la non-instanciation d'une variable
- En résumé :
  - disponibles dans tous les systèmes Prolog récents
  - permettent de retarder la démonstration d'une formule
  - utilisation dans la déclaration de contraintes passives
  - prolongement naturel : programmation par contraintes

UN LANGAGE DE PROGRAMMATION *LOGIQUE*

Prolog
III – listes et suites finies

Plan

1. Représentation
2. Propriétés
3. Accès aux éléments
4. Récursivité
5. Construction
6. Sous-suites

Plan

1. Représentation
2. Propriétés
3. Accès aux éléments
4. Récursivité
5. Construction
6. Sous-suites

Introduction

- Considérons l'exemple tiré du programme menu du TD1.
  - Avant : une règle associée à chaque élément du menu
    - entrée,
    - plat,
    - dessert,.
  - Maintenant : on va regrouper ces données et considérer
    - suite des entrées,
    - suite des plats,
    - suite des desserts.

Représentation (1)

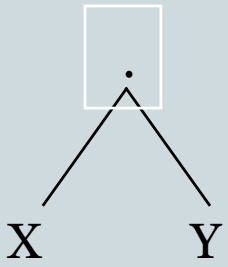
- A une suite, ordonnée ou non, on associe la liste de ses éléments.
  - Utilisation du symbole fonctionnel binaire « . »
    - suite {e1, e2, …} ➔ liste (e1.(e2.(…)))
  - Exemples :
    - suite des variables X et Y ➔ (X.Y)
    - suite {gateau, fruit, glace} ==>(gateau.(fruit.glace))

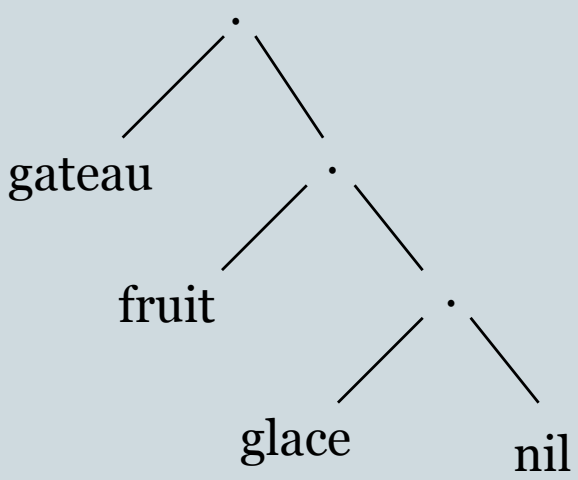
Représentation (2)

- La liste vide est notée « **nil** ».
- Elle sert souvent à marquer la fin de liste.
  - Il est préférable de noter la liste précédente sous la forme : *(gateau.(fruit.(glace.nil)))*
  - Cela définit le sens du parenthésage du symbole fonctionnel « . ».
  - Les listes peuvent alors être représentées sous forme d ’arbres.

Représentation (3)

- Exemples :





Représentation (4)

- Lorsque la liste comporte un trop grand nombre d'éléments, on admet le codage sans parenthèses
  - Exemples :
    - X.Y
    - gateau.fruit.glace.nil

Plan

1. Représentation
2. Propriétés
3. Accès aux éléments
4. Récursivité
5. Construction
6. Sous-suites

Propriété fondamentale (1)

- Jusqu'à présent ➔ arbres particuliers avec des branches gauches sans ramification.
  - On utilise le terme de **peigne** pour les désigner.

- Exercice :
  - résoudre l'équation *X.Y = gateau.fruit.glace.nil*
  - *Tapez : [X|Y] = [gateau, fruit, glace].*
  - par identification on a la solution :
    - {X = gateau; Y = [fruit, glace]}

Propriété fondamentale (2)

- La notation X.Y représente une liste dont
  - **tête** (le 1er élément) est X
  - **queue** (le reste de la liste) est Y.
    - Constitue la base de l'utilisation des listes dans les programmes Prolog.
    - **Attention** : considérez la liste *X.Y*.
      - Si Y ne se termine pas par **nil**, on ne connaît pas son type
        - *élément* si pas de nil à la fin,
        - *liste* si nil à la fin
      - d'où l'intérêt de **nil** pour s'assurer que Y est une liste.

Propriété fondamentale (3.a)

- Exemple du programme *menu* :

  entrees(sardine).
  entrees(pate).
  entrees(melon).
  entrees(avocat).

  - Et si on pose la question :

    entrees(E).

  - L'effacement de ce but (synonyme de question) provoque l'affectation suivante :

    E=sardine;
    E=pate;
    E=melon;
    E=avocat.

Propriété fondamentale (3.b)

- Exemple du programme *menu* :
  - *entrees([sardine, pate, melon, avocat]).*
  - Et si on pose la question :

    entrees(E).
  - L'effacement de ce but (synonyme de question) provoque l'affectation suivante :

    E=[sardine, pate, melon, avocat]

Propriété fondamentale (3.c)

- Complément du programme *menu* :

  viandes([poulet, roti, steack]).

  poissons([merlan, colin, loup]).

  desserts([gateau, fruit, glace]).

  - Et si on pose les questions :

    viandes(V).

    poissons(P).

    desserts(D).

Propriété fondamentale (4)

- Affecter une liste à une variable.
- Considérer les éléments d'une liste de façon individuelle
- Récupérer une entrée particulière.
  - Supposons que l'on dispose du prédicat :
  - *element-de(X, L)* capable d'extraire un objet X d'un peigne L.

Propriété fondamentale (5)

- Nous avions :

  menu(X, Y, Z) :- entree(X), plat(Y), dessert(Z).

  plat(X) :- viande(X).

  plat(X) :- poisson(X).

  - Avec la nouvelle déclaration commune des entrées comme liste, ce programme échouera. Il faut donc le modifier.
  - Pour cela nous allons construire une nouvelle définition du prédicat *entree*.

Propriété fondamentale (6)

- On utilise la propriété suivante : si X est une entrée, alors X est un élément quelconque de la liste des entrées.

  entree(X) :- entrees(E), element-de(X, E).

- Et de façon analogue :

  viande(X) :- viandes(V), element-de(X, V).

  poisson(X) :- poissons(P), element-de(X, P).

  dessert(X) :- desserts(D), element-de(X, D).

Plan

1. Représentation
2. Propriétés
3. Accès aux éléments
4. Récursivité
5. Construction
6. Sous-suites

Accès aux éléments d'une liste(1)

- Trouver règles assurant le fonctionnement du prédicat *element-de(X, L)* qui signifie que X est l'un des éléments de L.
  - Méthode de base mais trop particulière :
  - X est élément de L si
    - X est le premier élément,
    - ou le deuxième,
    - ...
    - ou le dernier.
  - Cela nécessite la connaissance de la longueur de la liste.

Accès aux éléments d'une liste(2)

- Autre méthode indépendante de la taille de L : utilise la remarque selon laquelle toute liste peut se décomposer simplement en 2 parties, la tête et la queue de liste. D'où deux cas :
  - X est élément de L si X est la tête de L.
  - X est élément de L si X est élément de la queue de L.

Accès aux éléments d'une liste(2)

- Ce qui se traduit directement en Prolog par :
  - R1 : *element-de(X, L) :- est-tete(X, L).*
  - R2 : *element-de(X, L) :- element-de-la-queue(X,L).*
  - R1 et R2 sont là en tant que repères et ne rentrent pas dans les règles.

Accès aux éléments d'une liste(3)

- Mais on peut aller plus loin car L peut toujours être représentée sous la forme *U.Y*, ce qui nous donne :
  - Si X est en tête de U.Y alors X = U.
  - Si X est un élément de la queue de U.Y alors X est élément de Y.
- D'où la version finale :
  - R1 : *element-de(X, [X|Y]).*
  - R2 : *element-de(X, [U|Y]) :- X\=U, element-de(X, Y).*

Accès aux éléments d'une liste(4)

- Commentaires :
  - On interprète ces deux règles de la façon suivante :
    - R1 : X est élément de toute liste qui commence par X.
    - R2 : X est élément de toute liste dont la queue est Y, si X est élément de Y.
  - Le second membre de R2 provoque un nouvel appel à R1 et R2. Il s'agit donc d'un appel récursif.
  - La plupart des problèmes sur les listes ont des solutions mettant en jeu la récursivité.

Plan

1. Représentation
2. Propriétés
3. Accès aux éléments
4. Récursivité
5. Construction
6. Sous-suites

Récursivité (1)

- On observe deux types de règles et deux types d'arrêt :
  - Une ou plusieurs règles provoquent la récursivité, généralement sur des données assez simples, et assurent le déroulement de la boucle. Dans notre exemple, il s'agit de la règle **R2**.
  - Une ou plusieurs règles stoppent la boucle. Dans notre exemple, c'est **R1** qui s'en charge.
  - Sauf impératif contraire, les règles d'arrêt sont placées en tête.

Récursivité (2)

- Conseils pour l'écriture de programmes récursifs :
  - Chaque règle doit être, autant que possible, conçue comme une déclaration ou une définition, indépendamment de toute forme d'exécution.
  - L'ordre des règles peut être déterminé par l'exécution du programme (et pas seulement pour les parties récursives du programme).

Récursivité (3)

- Il apparaît deux types d'arrêt :
  - Un arrêt explicite. Par exemple, dans R1, l'identité entre l'élément cherché et la tête de la liste fournie, ce qu'exprime la notation *X* et *X.Y*.
  - Un arrêt implicite. En particulier par la rencontre d'un terme impossible à effacer.
  - Il existe cependant une forme d'erreur pour laquelle ces deux blocages se révèlent insuffisants, c'est la rencontre de listes infinies.

Plan

1. Représentation
2. Propriétés
3. Accès aux éléments
4. Récursivité
5. Construction
6. Sous-suites

Construction d'une liste (1)

- Nous avons vu comment parcourir une liste.
- Nous allons voir comment en construire une.
  - Examinons le problème suivant : L une liste contenant un nombre pair d'éléments.
    - Par exemple : L = 0.1.2.3.4.5.nil
  - Cherchons à construire une nouvelle liste W en prenant les éléments de rang impair dans L.

Construction d'une liste (2)

- Dans notre exemple cela donnerait :
  - W = 0.2.4.nil
- Nous disposons de deux outils : le découpage d'une liste et la récursivité. Méthode à suivre :
  - Création d'un appel récursif.
  - Modification des règles pour obtenir le résultat désiré.

elements-impairs(L) :- queue(L, Y), queue(Y, Y1),
elements-impairs(Y1).

Construction d'une liste (3)

- On a L = X1.Y ce qui nous donne :

elements-impairs(X1.Y) :- queue(X1.Y, Y),
queue(Y, Y1), elements-impairs(Y1).

- La première condition est alors inutile. De plus la décomposition de L peut etrerepetee avec Y = X2.Y1. On a donc :

elements-impairs(X1.X2.Y1) :- elements-impairs(Y1).

- Nous avons donc notre règle récursive. Il reste à trouver la règle d'arrêt. Il faut s'arrêter à la liste vide :

elements-impairs(nil).

Construction d'une liste (4)

- D'où le programme :

  R1 : elements-impairs(nil).

  R2 : elements-impairs(U.V.Y) :- elements-impairs(Y).

- Il reste à modifier le programme de façon à construire la nouvelle liste à partir du parcours de l'ancienne. D'où le programme final :

  R1 : elements-impairs(nil, nil).

  R2 : elements-impairs(U.V.Y, U.M) :- elements-impairs(Y, M).

Construction d'une liste (5)

- Interprétation de ces deux règles :
  - R1 : prendre un élément sur deux dans la liste vide donne la liste vide
  - R2 : prendre un élément sur deux dans la liste U.V.Y donne la liste U.M si prendre un élément sur deux dans la liste Y donne la liste M.
- Remarque : les éléments dans la liste résultat sont rangés dans le même ordre que dans la liste initiale.

Construction d'une liste (6)

- Construction d'une liste au moyen d'une liste auxiliaire :
  - Problème : soit D une liste donnée, R est la liste D inversée. On est dans une situation différente la précédente à cause de l'ordre inverse des éléments. Nous allons donc utiliser une liste auxiliaire pour ranger les éléments au fur et à mesure de leur lecture en attendant de pouvoir les utiliser.

Construction d'une liste (7)

- Boucle de récursivité :

  renverser(D) :- tete(X, D), queue(Y, D), renverser(Y).

  - Comme précédemment, on remplace D par X.Y ce qui donne :

  renverser(X.Y) :- renverser(Y).

  renverser(nil).

- Nous allons maintenant ajouter deux arguments à ces règles, la liste auxiliaire et la liste résultat R.

Construction d'une liste (8)

- On obtient les règles suivantes :

  R2 : *renverser(D, L, R) :- renverser(Y, X.L, R).*

  R1 : *renverser(nil, L, R) :- transformer(L, R).*

- Il faut déterminer la transformation à effectuer sur L pour obtenir R.
  - Au lancement L est vide.
  - Chaque fois qu'un élément apparaît en tête de D, il est retiré et placé en tête de L. Il y a donc inversion de l'ordre des éléments donc L et R sont identiques. Il n'y a pas de transformation à effectuer.

Construction d'une liste (9)

- Le programme final est donc :

  R1 : *renverser(nil, L, L).*

  R2 : *renverser(X.Y, L, R) :- renverser(Y, X.L, R).*

- Interprétation de ces deux règles :
  - Renverser la liste vide, à partir de la liste auxiliaire L, donne la liste résultat L.
  - Renverser la liste X.Y, à partir de la liste auxiliaire L, donne la liste résultat R, si renverser la liste Y, à partir de la liste auxiliaire X.L, donne la même liste R.

Construction d'une liste (10)

- Exercice classique : concaténation de deux listes
  - Plusieurs possibilités :
    - récursivité
    - découpage de la liste
    - construction directe
    - utilisation d 'une liste auxiliaire
    - passage final de paramètre (méthode précédente)

Plan

1. Représentation
2. Propriétés
3. Accès aux éléments
4. Récursivité
5. Construction
6. Sous-suites

Listes et sous-suites (1)

- Utilisation du programme **menu** :
  - Volonté de poursuivre le regroupement des données.
  - Liste unique qui regrouperait tous les mets du menu
    - L= sardine.melon…glace.nil
  - Cette représentation n'est pas adéquate car il n'y a pas de distinction entre entrée, plat ou dessert.
  - Il est donc nécessaire de découper la liste en sous-listes qui rassembleront les entrées, plats et desserts. La sous-liste des plats sera elle-même divisée en deux parties, les viandes et les poissons.

Listes et sous-suites (2)

- Cela nous donne la structure :

L = (sardines.pate.melon.celeri.nil).((poulet.roti.steack.nil).(merlan.colin.loup.nil)).(gateau.fruit.glace.nil).nil

L est de la forme : L = L1.L2.L3.nil

- Exercices :
  - Construire l'arbre associé à la liste L.
  - Résoudre les équations : L=X.Y, L=X.Y.Z, L=X.Y.Z.T, L=X.(U.V).Y, L=(X.Y).Z, L=X.((U.V).Y).Z, L=X.(U.V).Y.Z

FIN PREMIERE PARTIE

blablablablablablablablabla‘blablablablabla’

« J’ai écrit un article intitulé "**le web sémantique n’est pas antisocial**"»

FIN PREMIERE PARTIE