Représentation des connaissances

Isis TRUCK

Université Paris 8

2013 – 2014

Bibliographie

- D. Kayser, *La représentation des connaissances*, Hermès, 1997.
- D. Vernant, *Introduction à la logique standard, Calcul des propositions, des prédicats et des relations*, Flammarion, 2011.
- B. Bouchon-Meunier, *La logique floue et ses applications*, Addison-Wesley, 1995.
- N. Sabouret, *Représentation des connaissances en XML. Le Web sémantique*, LIP6, 2006.

Plan

Plan

Connaissances : qu'est-ce que c'est ?

- des faits,
- des propositions,
- des suppositions,
- des croyances,
- des hypothèses,
- des idées,
- des savoirs,
- mais aussi des savoir-faire, des techniques, des heuristiques, etc.

Représenter, pour quoi faire ?

- pour raisonner,
- pour inférer de la connaissance

Attention !

Inférence $\neq$ raisonnement, mais raisonnement $\in$ inférence

Nota Bene

L'inférence est l'élément de base de toute description de la cognition.

Pour être efficace, l'inférence doit être guidée par la connaissance.

Automatisation ?

- $\nexists$ machine universelle pour réaliser n'importe quelle tâche complexe
- au mieux, machine peut réaliser tâche complexe *localement* $\Rightarrow$ donc il faut des connaissances *sur le domaine*
- $\Rightarrow$ comment *communiquer* les connaissances à la machine ?

Nota Bene

Les représentations doivent servir de *condensé d'expérience*.

Plan

Définition

- le *codage* n'est qu'une partie du problème de la représentation
- la représentation, quant à elle, implique un mécanisme pour *guider* l'inférence

Nota Bene

Le codage n'est qu'un simple stockage *condensé* et *fiable*

Comment coder ?

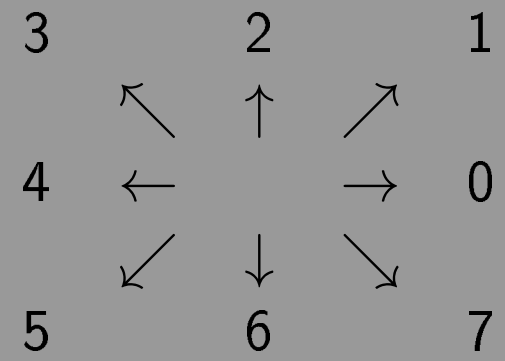
- pour lever les ambiguïtés, codage = symbole + type
- exemple : <“four”; number>
- symboles $\in$ alphabet. Exemples :
  - 26 lettres $\{a, \ldots, z\}$
  - ens. de notes $\{A, B, C, D, E\}$
  - $\mathbb{R}, \mathbb{Z} \ldots$
  - $\{0, 1\}$: binaire est suffisant ! Pourquoi ?
- longueur du codage fixe ou variable (*cf.* TD 1)

Codage des nombres

- nécessité d'utiliser des "astuces" pour simplifier les opérations usuelles
- $\Rightarrow$ codage en binaire avec bit de signe (complément à 2, *cf.* TD 1)
- nombres flottants : mantisse m et exposant e
  - $\Rightarrow r = m.2^e$
  - question de la *granularité* : écart relatif maximal entre 2 réels non nuls dont le codage est identique
  - choix de m et e, sachant que nb de cases pour stocker m et e est borné ?
- il faut connaître le codage pour décoder. Ex. : 10001000 : quelles interprétations ?

Codage des images

- problème de la réduction du continu au discret (*cf.* granularité)
  $\Rightarrow$ échantillonnage, perte d'information...
- utilisation d'un maillage avec un point de départ et une liste de valeurs indiquant des directions :



- $\Rightarrow$ mais cette méthode ne code que les contours continus
- utilisation d'un maillage de pixels, chaque pixel étant codé par son niveau de gris (sur 8 bits), ou par sa couleur (espace colorimétrique RGB, HLS, etc. utilisant 3 fois 8 bits, un octet par composante)

Codage *vs.* recherche d'information

Le codage

Il associe à une suite d'éléments d'un même type une *suite de symboles* (un code).

La recherche d'information

Elle associe à un code d'un certain type et à un élément du même type un *booléen* correspondant à l'existence ou non de l'élément dans le code, une fois l'élément traduit en symbole.

Recherche d'une information ?

- plusieurs types de recherche :
  - **séquentielle** : les n symboles ne sont pas supposés "rangés". Au pire, n comparaisons à effectuer ; en moyenne, $(n+1)/2$
  - **dichotomique** : présuppose un tri. Si $n = 2^k - 1$, au pire : k comparaisons à effectuer ; en moyenne, $k - 1$
  - **directe** : symboles sont stockés dans un tableau de taille `nbTotalElmts`. Si un 1 est à la place de l'élément recherché, alors l'élément est bien présent. Dans tous les cas, une seule comparaison à effectuer
  - **adressage dispersé** : symboles ne sont plus des booléens mais codent une information supplémentaire (par exemple, le résultat d'un modulo).
    $\Rightarrow$ Permet des temps d'accès **indépendants de la quantité d'information stockée**.

Explication

- Soit m le facteur de surdimensionnement du tableau, c'est-à-dire le rapport entre le nombre de cases et le nombre n d'éléments de l'ensemble à coder. p est le nombre de positions utilisées pour coder les éléments. On peut montrer, moyennant des hypothèses raisonnables, que les caractéristiques sont :
  - place occupée par le code : mnp
  - nombre de comparaisons à effectuer :
    - en cas de succès : $m \ln \frac{m}{m-1}$
    - en cas d'échec : $\frac{m}{m-1}$
- $\Rightarrow$ Le nombre de comparaisons à effectuer (temps d'accès) ne dépend pas de n

Notions de **clé** et de **tolérance**

- recherche d'information dont on ne connaît qu'*une partie* ?
- $\Rightarrow$ si partie suffisante pour identifier, alors c'est une **clé** (*cf.* SGF (i-node), BD (tables liées)...)

- recherche d'information avec une certaine tolérance :
  - si distorsions obéissent à principe connu, utilisation de *classes d'équivalence* pour les gommer. Ex. : casse (maj/min), variante orthographique (Istambul, Istanbul, Istamboul)
  - sinon, recherche **approximative** (*cf.* calculs de distance, logique floue, réseaux de neurones, etc.)

Guider les inférences ?

Code *vs.* connaissance

Les codes sont *passifs* alors que l'on les souhaitait *actifs*.

Il n'y a présomption de *connaissance* que si la faculté d'**utiliser** des informations **à bon escient** est attestée.

- il y a donc un lien entre *connaître* et être susceptible d'agir
- les informations sont *exploitées* par des psus sans pouvoir modifier leur déroulement, tandis que
- les connaissances sont des données qui *influencent* le déroulement du psus

Les différentes natures de connaissances

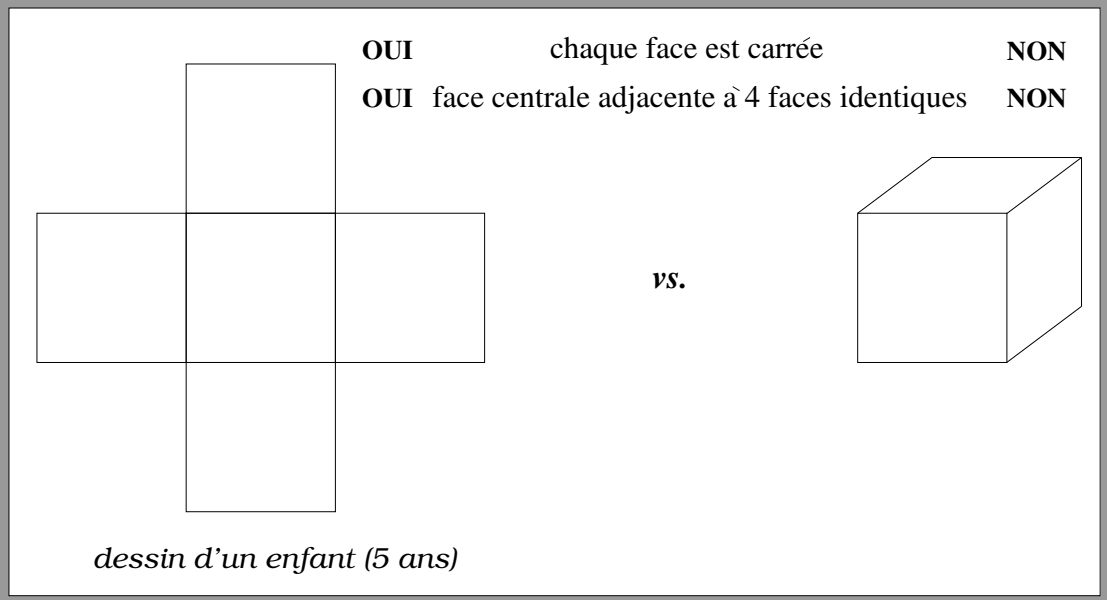
- connaissances de type mode d'emploi (*cf.* métaconnaissances)
- connaissances de définition ("avion : véhicule qui vole")
- connaissances évolutives ("1^{er} ministre : François Fillon")
- connaissances incertaines ("il viendra peut-être demain")
- connaissances vagues ("adolescents : jeunes de 12 à 16 ans")
- connaissances typiques ("un enfant passe 4 ans au collège")
- connaissances sous-déterminées ("les personnes ayant 3 enfants ont une réduction" $\Rightarrow$ 3 signifie en fait "≥ 3")

$\Rightarrow$ choix de catégorie dépend du point de vue
$\Rightarrow$ caractère déterminé ou non dépend du degré de précision (bcp d'ambiguïtés sur '3' : 3 enfants vivants ? 3 enfants mineurs ? etc.)

Plan

Généralités (1)

- Aucune représentation n'a de valeur indépendamment de son utilisation : pas possible de séparer l'étude des techniques de représentation de celle des techniques d'exploitation de cette représentation
- Ex. cube [Minsky, 1972] :



Généralités (2)

Nota Bene

Une représentation est toujours une approximation.

- Si la représentation possède *toutes* les propriétés de l'entité, c'est l'entité elle-même !
- Représenter suppose de sacrifier certaines propriétés
- Ex. carte routière :
  - si échelle 1, alors ce n'est plus une carte !
  - une bonne carte est une carte qui *omet* les éléments inutiles (et perturbants)

Nota Bene

$\nexists$ représentation qui soit uniformément "meilleure" qu'une autre.

Modèle

- on modélise le monde grâce aux informations codées par des symboles
- les représentations sont des *structures de symboles*

Nota Bene

Interpréter présuppose que le modèle est constitué d'*objets* et que parmi les symboles, il en est qui s'interprètent comme des *objets*.

Les symboles ont la capacité de déclencher des inférences.

- Ex. avec arbres et NPI : les feuilles sont des opérandes et les nœuds des opérateurs

Plan

Langage de représentation

- Représentations = séquences de symboles obéissant à des règles
- $\Rightarrow$ **langage** des représentations (langages *artificiels*)

Définitions

Alphabet : ensemble fini Σ de symboles

Symbole = 1 signe $(\mathrm{a}, 4 \ldots)$ ou plusieurs signes ($\underline{\mathrm{debut}}$, $\underline{\mathrm{mot}}$...)

Σ^* : ens. de toutes les suites qu'on peut construire en juxtaposant des symboles de Σ

ε : suite vide (formée de zéro symbole)

Langage : sous-ens. quelconque de Σ^*

Langage de représentation : réécriture

- Utilisation d'un **alphabet auxiliaire** *Aux* pour caractériser les langages par réécriture
- **Réécriture** : $\mathrm{a} \rightarrow \alpha$, où $\mathrm{a} \in Aux$ et $\alpha \in \Sigma \cup Aux$
- **Axiome** : un élt de *Aux* qui sert de point de départ à toute dérivation (par réécriture)
- A partir de l'axiome, on doit pouvoir **engendrer** grâce aux réécritures **toutes les suites de symboles** que l'on souhaite représenter
- Une **séquence** est une suite de réécritures
- $\Rightarrow$ la 1^{re} suite d'une séquence est toujours l'axiome

Langage de représentation : exemple (*cf.* TD 1)

- On pose : $\Sigma = \{\mathrm{a}, \mathrm{b}\}$; *Aux*$= \{\underline{\text{phrase}}\}$
- Axiome : $\underline{\text{phrase}}$
- On impose les deux réécritures suivantes :
  1. $\underline{\text{phrase}} \rightarrow \text{a a } \underline{\text{phrase}}$
  2. $\underline{\text{phrase}} \rightarrow \text{b}$

- Quelle séquence permet d'obtenir
  - a a a a b ?
  - b b $\underline{\text{phrase}}$?

- *Nota Bene* : les 2 réécritures peuvent s'écrire :
  $\underline{\text{phrase}} \rightarrow \text{a a } \underline{\text{phrase}} \mid \text{b}$

Ambiguïtés

- Lorsqu'une suite peut provenir de plus d'une séquence (au moins 2 dérivations distinctes génèrent la même suite), la description proposée par les réécritures est dite *ambigüe*. **Donner un exemple**.
- Pour éviter cela, on impose des *contraintes*, mais cela enserre du coup beaucoup le langage. C'est d'ailleurs une des raisons pour lesquelles le langage naturel ne peut être modélisé de la sorte.
- De même, modéliser le Code Civil est très fastidieux pour les mêmes raisons (le Code Civil n'est pas un *langage* contraint, c'est une suite de prédicats qui sont parfois vagues, qui peuvent se contredire ou s'interpréter de plusieurs manières)

Plan

Représentation de procédures (1)

- *Nota Bene* : Sachant que les K peuvent elles-mêmes être procédurales, représenter les procédures $\in$ représentation des K

- Reprenons les langages de représentation en les adaptant pour les procédures
- $\Rightarrow$ il suffit d'ajouter aux alphabets (Σ et *Aux*) des symboles représentant des actions, par exemple,
  - debut
  - fin
  - *instruction*
  - *condition*
  - etc.

Représentation de procédures (2)

- Exemple. Soient $\Sigma = \{\mathrm{a}, \mathrm{b}, \ldots, \mathrm{z}\} \cup \{;, >, <, 0, \underline{\text{debut}}, \underline{\text{fin}}, \underline{\text{tantque}}, \underline{+\text{un}}, \underline{-\text{un}}, \underline{=\text{zero}}\}$;
  $Aux = \{\underline{procedure}, \underline{instruction}, \underline{nom}, \underline{condition}\}$;
  Axiome : $\underline{procedure}$
- Les réécritures sont :
  1. $\underline{procedure} \rightarrow \underline{instruction}\ ;\ \underline{procedure} \mid \underline{instruction}$
  2. $\underline{instruction} \rightarrow \underline{=\text{zero}}\ \underline{nom} \mid \underline{+\text{un}}\ \underline{nom} \mid \underline{-\text{un}}\ \underline{nom} \mid \underline{\text{tantque}}\ \underline{condition}\ \underline{\text{debut}}\ \underline{procedure}\ \underline{\text{fin}}$
  3. $\underline{nom} \rightarrow \mathrm{a}\ \underline{nom} \mid \ldots \mid \mathrm{z}\ \underline{nom} \mid \mathrm{a} \mid \ldots \mid \mathrm{z}$
  4. $\underline{condition} \rightarrow \underline{nom} > 0 \mid \underline{nom} < 0$

Représentation de procédures (3)

- Exercice :
  - Que fait la suite (désormais appelée **procédure**) suivante ?
    $\underline{=\text{zero}}$ y ; $\underline{\text{tantque}}$ x > 0 $\underline{\text{debut}}$ $\underline{-\text{un}}$ x ; $\underline{+\text{un}}$ y ; $\underline{+\text{un}}$ y $\underline{\text{fin}}$
    avec $\text{x} \geq 0$
  - Représenter sous forme d'arbre la séquence qui permet d'arriver à cette procédure.

Procédures vues comme des objets (1)

- En considérant qu'une procédure peut être la *donnée* d'une autre procédure, on obtient des données qui représentent des procédures **actives**, et non plus seulement des objets passifs comme précédemment
- Ceci a donné lieu au formalisme du λ-calcul [Church, 1951], à la base du langage LISP [McCarthy, 1959] et qui est une **représentation formelle de procédures**
- Exemple : fonction successeur $(+1)$ s'écrit $\lambda x \bullet x + 1$.
- L'intérêt de ce formalisme est que ses expressions peuvent s'appliquer à des arguments qui sont eux-mêmes des fonctions (*cf.* TD 1)

Procédures vues comme des objets (2)

- D'autres représentations formelles de procédures existent : système canonique de Post, machine de Turing, etc.
- Machine de Turing, définition [Turing, 1936] :

Définition 1.3.1 (Machine de Turing) *Une machine de Turing consiste en:*
- un ruban, avec une extrémité à gauche, infini à droite, divisé en cases de même taille. On peut penser par exemple à une bande magnétique suffisamment longue;
- un ensemble fini de symboles, par exemple $\{0,1,s,d,f\}$*, avec* 0 *et* 1 *pour la numérotation binaire, s pour séparer deux expressions, d pour le début du ruban et f pour signifier que ce qui est à droite n'a plus d'importance. Ils seront inscrits et lus sur le ruban à raison d'un symbole par case;*
- une tête de lecture/écriture qui se déplace sur le ruban. A chaque impulsion, la tête peut soit rester sur place, soit reculer (si possible) ou avancer, dans les deux cas d'une case. La tête a le droit de lire ou d'écrire dans la case qu'elle est en train de sonder. Pour rester concret, on peut penser que c'est la bande et non la tête qui bouge;
- un ensemble fini d'états. Ces états permettent de distinguer plusieurs comportements possibles. Notamment l'état D *représente l'état de départ et l'état* F *symbolise la fin du "programme" et donc la lecture du résultat par un observateur extérieur;*
- un ensemble fini d'instructions: à chaque impulsion, en fonction du symbole c *lu par la tête, en fonction de l'état courant* S*, la tête écrit un nouveau symbole* c'*, effectue un déplacement noté* -1 *(gauche),* $+1$ *(droite) ou* 0 *(immobile) et passe à l'état* S'*.*

Procédures vues comme des objets (3)

- On a prouvé que chacune de ces représentations capture la *même* notion de procédure

Thèse de Church, 1943

Tout procédé de calcul est représentable à l'aide d'une de ces représentations car la notion de *méthode effective de calcul* y est correctement formalisée.

- Cependant, il n'existe pas de procédé de calcul indiquant, pour toute procédure et pour toute donnée, si la procédure termine ou non $\Rightarrow$ **indécidabilité** du **problème de l'arrêt**

Procédures vues comme des objets (4)

- Pour autant, même s'il n'existe pas de procédé répondant dans **tous** les cas si une procédure s'arrête ou non, il existe bien sûr dans **certains** cas des procédés qui répondent à la question
- Rappel : un problème de décision est dit **décidable** s'il existe un algorithme (ou une procédure qui termine en un nombre fini d'étapes), qui le décide, c'est-à-dire qui réponde par oui ou par non à la question posée par le problème.
- Exemple : soit la procédure suivante :
  $\underline{=\text{zero}}\ \text{x}\ ;\underline{+\text{un}}\ \text{x}\ ;\underline{\text{tantque}}\ \text{x} > 0\ \underline{\text{debut}}\ \underline{-\text{un}}\ \text{x}\ ;\ \underline{+\text{un}}\ \text{x}\ ;\ \ \underline{\text{fin}}$
  Est-ce que cette procédure termine ?

Plan

Des modèles qui sont des *logiques*

- Des formalismes ont vu le jour pour à la fois représenter, mais également raisonner sur les connaissances $\Rightarrow$ systèmes déductifs
- Ces systèmes formels sont appelés des **logiques**
- Ils cherchent à modéliser *certaines* connaissances et *certains* comportements
- Une logique comporte : un langage (*cf.* infra), un système de déduction et des règles de valuation (qui donne une valeur à toute formule du langage)

Plan

Les logiques (1)

- La logique des propositions (*cf.* cours à part)
- Les logiques multivaluées
  - Logique trivalente de Łukasiewicz
  - Logique floue (*cf.* cours à part)

Les logiques (2)

- La logique des propositions
  - Elle est décidable
  - mais elle est peu expressive !
  - et temps de décision exponentiels
- Les logiques multivaluées $\Rightarrow$ décidables ou semi-décidables mais temps de décision exponentiels
  - Logique trivalente de Łukasiewicz
  - Logique floue
- Autre chose pour représenter les connaissances ?

Plan

Les réseaux sémantiques (1)

- Il faut restreindre la logique pour avoir des algorithmes complets et rapides
- Il faut abandonner l'exigence de **complétude**

Nota Bene

Il y a complétude si toutes les formules qui sont des tautologies sont également des théorèmes. Ainsi, un système de déduction est complet quand toutes les déductions sémantiques peuvent se dériver dans le système. La logique des propositions est complète.

Les réseaux sémantiques (2)

- Idée : proximité spatiale et proximité sémantique sont liées
- Réseaux sémantiques [Doyle, 1962] [Quillian, 1968]
- Notion de nœuds, de relations, d'héritage, etc.
- Expression graphique des relations

Les réseaux sémantiques (3)

- Réseau sémantique : graphe orienté et étiqueté composé de nœuds reliés entre eux par des arcs
- Les arcs n'ont pas tous la même signification (AKO pour *a kind of* ; *is-a* ; *part-of* ; *attribute* ; *is* ; *has* ; *can*)
- Les nœuds n'ont pas tous même signification (ils peuvent représenter des classes d'individus, des individus, des attributs, etc.)
- Parmi les réseaux sémantiques, il existe les **logiques de description** [Brachman & Levesque, 1984], (*cf.* cours à part pour ceux qui choisiront ce sujet en devoir à la maison)

Plan

Conclusion

- Richesse expressive implique une grande complexité
- La lecture des modèles est souvent, voire toujours difficile
- Les mécanismes de calcul (projection, subsomption, etc.) ne permettent pas des expressions de requêtes toujours simples (càd qu'il faut reformuler).