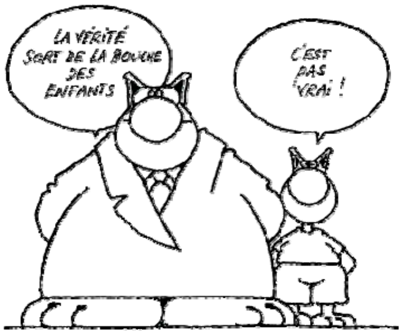
Sunday, February 25, 2018

SD 206

Logic and Knowledge representation

Jean-Louis Dessalles

Dep. InfRes



Voir aussi la bande dessinée:
http://www.logicomix.com/fr/

prolog

History

http://individual.utoronto.ca/pking/miscellaneous/history-of-logic.pdf

- Ancient greeks
  - Stoics
  - Aristotle
    - Syllogism
    - Argumentation
- Medieval logic
  - William of Ockham (1288-1348)
    - de Morgan's laws
    - Ternary logic
- Traditional logic
  - Port Royal's logic
    - Antoine Arnauld & Pierre Nicole (1662)
      - Logic of propositions
- Modern Logic
  - Descartes, Leibniz
  - George Boole (1848)
  - Gottlob Frege
    - *Bregriffschrift* (1879)
      - *Quantification*
  - Charles Peirce
  - Guiseppe Peano
    - Logical axiomatization of artihmétics
  - Bertand Russell & Alfred N. Whitehead (1925)
    - Logical axiomatization of mathematics

prolog

All Bankers are Athletes,
No Consultant is a Banker.
So…

Some athletes aren't consultants.

prolog

Syntaxe

Alphabet

symboles propositionnels : p1 , p2 , ...

connecteurs

à 0 place : T, $\bot$ (constantes)

à 1 place : $\neg$

à 2 places : $\wedge$ $\vee$ $\supset$ $\subset$ $\uparrow$ $\downarrow$ $\not\subset$ $\not\supset$

Formule Atomique

si F est une constante, alors F $\in$ ***A***

si F est un symbole propositionnel, alors F $\in$ ***A***

Formule Propositionnelle

si F est une formule atomique, alors F $\in$ ***F***

si F $\in$ ***F***, alors $\neg F$ $\in$ ***F***

si $\bullet$ est un connecteur à deux places, si $F_1 \in$ ***F*** et $F_2 \in$ ***F***, alors $(F_1 \bullet F_2) \in$ ***F***

F est le plus petit ensemble possédant ces propriétés

prolog

Truth values

Valeurs de Vérité

Vr = {v,f}

Fonctions de Vérité

à 0 place : v,f

à 1 place : ***Vr*** $\rightarrow$ ***Vr***

	Non
V	F
F	V

à 2 places : ***Vr*** $\times$ ***Vr*** $\rightarrow$ ***Vr***

		Et	Ou	$\Rightarrow$	$\Leftarrow$	net	nou	$\nRightarrow$	$\nLeftarrow$
V	**V**	V	V	V	V	F	F	F	F
V	**F**	F	V	F	V	V	F	V	F
F	**V**	F	V	V	F	V	F	F	V
F	**F**	F	F	V	V	V	V	F	F

prolog

Sémantique (Truth values)

Evaluation Booléenne

$\nu : F \to Vr$

$\nu\ (F) \in \{v,f\}$

$\nu\ (T) = v$

$\nu\ (\bot) = f$

$\nu\ (\neg F) = \text{non}\ \nu\ (F)$

$\nu\ ((F_1 \bullet F_2)) = \nu\ (F_1) \blacklozenge \nu\ (F_2)$

Connecteur syntaxique $\bullet$	Connecteur sémantique $\blacklozenge$
$\neg$	Non
$\wedge$	et
$\vee$	Ou
$\supset$	$\Rightarrow$
$\subset$	$\Leftarrow$
$\uparrow$	Net
$\downarrow$	Nou
$\not\supset$	$\nRightarrow$
$\not\subset$	$\nLeftarrow$

prolog

Logical consequence

•A propositional formula *X* is a *tautology*
if $v(X) = \boldsymbol{T}$ for any Boolean valuation *v*.

•A set *S* of propositional formulas is *satisfiable*
if some valuation v_0 maps every member of *S* to *T*:
$v_0(X) = \boldsymbol{T}$ for any *X* of *S*.

•One can easily verify that *X* is a tautology if and only if $\{\neg X\}$ is not satisfiable.

•Let's note $S \vDash X$ the *propositional consequence*:
if a valuation assigns the value ***T*** to any element of *S*, then it will assign ***T*** to *X*.

prolog

Logical consequence

$S \models X$

if a valuation assigns the value ***T*** to any element of S, then it will assign ***T*** to X.

$\models X$

X is a tautology.

Show that if $S \models X$, then $S \cup \{\neg X\}$ is not satisfiable.
Show the reciprocal (refutation)

(*ex falso quodlibet sequitur*).
Show that if $A, \neg A \in S$, then for any X : $S \models X$.
Conversely, if for any X : $S \models X$, then show that S is not satisfiable.

(monotony).
Show that if $S \models X$ then $S \cup \{Y\} \models X$

(deduction).
Show that $S \cup \{X\} \models Y$ iff $S \models (X \supset Y)$

Show that $(\neg(X \wedge Y) \equiv (\neg X \vee \neg Y))$ is a tautology.

Show that X is a tautology iff
$(X \equiv \mathrm{T})$ is a tautology, and iff
$(\mathrm{T} \supset X)$ is a tautology.

Replacement Theorem

$F(P)$ is a propositional formula with zero, one or several occurrences of symbol P

Replacement Theorem

if $(X \equiv Y)$ is a tautology, then $(F(X) \equiv F(Y))$ is a tautology as well.

Replacement Theorem

On note F(P) une formule propositionnelle avec zéro, une ou plusieurs occurrences de la formule P.

Pour F(P), P_1 et P_2 des formules propositionnelles, et $\mathbf{v}$ une évaluation booléenne,

si $\mathbf{v}$ $(P_1) = \mathbf{v}$ (P_2) alors $\mathbf{v}$ $(F(P_1)) = \mathbf{v}$ $(F(P_2))$.

On définit le connecteur d'équivalence $\equiv$, tel que pour P_1 et P_2 des formules propositionnelles,

$\mathbf{v}$ $((F_1 \equiv F_2)) = v$ ssi $\mathbf{v}$ $(F_1) = \mathbf{v}$ (F_2).

On démontre que pour P_1 et P_2 des formules propositionnelles, $(P_1 \equiv P_2)$ est une tautologie ssi $\mathbf{v}$ $(P_1) = \mathbf{v}$ (P_2) pour toute évaluation booléenne $\mathbf{v}$, et ssi $\{P_1\} \models P_2$ et $\{P_2\} \models P_1$.

Si $(P_1 \equiv P_2)$ est une tautologie, alors $(F(P_1) \equiv F(P_2))$ est une tautologie.

prolog

$(\neg\neg X \equiv X)$ is a tautology.

$(\neg\neg X \supset p)$ et $(X \supset p)$ have same "semantics"

(*modus ponens*)
Show that Y is a tautology
if X and $(X \supset Y)$ are tautologies

prolog

Forme normale pour la négation

Une formule propositionnelle est sous forme normale pour la négation si les symboles de négation n'apparaissent que devant les symboles propositionnels.

Il existe un algorithme qui convertit une formule propositionnelle X en une formule propositionnelle Y qui est sous forme normale pour la négation, telle que $(X \equiv Y)$ soit une tautologie.

Les premières étapes d'un tel algorithme consiste à faire les transformations suivantes pour toute formule F, sous formule de X :

si F est $\neg\perp$ alors la remplacer par T.
si F est $\neg$T alors la remplacer par $\perp$.
si F est de forme $\neg\neg$P alors la remplacer par P.

prolog

Disjonction, conjonction généralisées

Soient $X_1, X_2, \ldots, X_n$ des formules propositionnelles :

la formule $[X_1, X_2, \ldots, X_n]$ est la disjonction généralisée de $X_1, X_2, \ldots, X_n$.
la formule $< X_1, X_2, \ldots, X_n >$ est la conjonction généralisée de $X_1, X_2, \ldots, X_n$.

Soit ν une évaluation booléenne :

$\nu([X_1, X_2, \ldots, X_n]) = f$ ssi $\nu(X_i) = f$ pour tout i.
$\nu(<X_1, X_2, \ldots, X_n>) = v$ ssi $\nu(X_i) = v$ pour tout i.

$\nu([\,]) = f$ $([\,] \equiv \perp)$
$\nu(<>) = v$ $(<> \equiv T)$

prolog

Conjunctive Normal form

The *cunjunctive normal form* (CNF) rewrites any propositional formula as a conjunction of *clauses*;
each of these clause is a generalized disjunction of propositional symbols possibly with negation.

A clause is noted [*a, b, c*].
A conjunction of clauses is noted $< C_1, C_2, C_3 >$.
A propositional formula is *in CNF* if it is written as a conjunction of clauses : $< C_1, C_2, \ldots C_n >$ where each C_i is a clause.

$v([\, X_1, X_2, \ldots X_n \,]) = \boldsymbol{F}$ iff $v(X_i) = \boldsymbol{F}$ for all i
$v(< C_1, C_2, \ldots C_m >) = \boldsymbol{T}$ iff $v(C_i) = \boldsymbol{T}$ for all i

$v([\,]) = \boldsymbol{F}$, $v(<>) = \boldsymbol{T}$.

prolog

Formes normales disjonctives, conjonctives

Une formule propositionnelle est sous forme *normale disjonctive* si elle s'écrit comme une disjonction généralisée [C1, C2, …, Cn] où chaque Ci est une conjonction généralisée.

Une formule propositionnelle est sous forme *normale conjonctive* si elle s'écrit comme une conjonction généralisée <D1, D2, …, Dn> où chaque Di est disjonction généralisée.

Il existe des algorithmes qui convertissent toute formule propositionnelle X en une formule Y qui est sous forme normale (conjonctive ou disjonctive), telle que (X ≡Y) est une tautologie.

prolog

Remplacing primitive connectives

Pour toute valuation booléenne v, $v(\alpha) = (v(\alpha_1)$ et $v(\alpha_2))$, et $v(\beta) = (v(\beta_1)$ ou $v(\beta_2))$.

Pour chaque formule α ou β, $\alpha \equiv (\alpha_1 \wedge \alpha_2)$, $\beta \equiv (\beta_1 \vee \beta_2)$ sont des tautologies.

α	α_1	α_2
$(X \wedge Y)$	X	Y
$\neg(X \vee Y)$	$\neg X$	$\neg Y$
$\neg(X \supset Y)$	X	$\neg Y$
$\neg(X \subset Y)$	$\neg X$	Y
$\neg(X \uparrow Y)$	X	Y
$(X \downarrow Y)$	$\neg X$	$\neg Y$
$(X \not\supset Y)$	X	$\neg Y$
$(X \not\subset Y)$	$\neg X$	Y

β	β_1	β_2
$\neg(X \wedge Y)$	$\neg X$	$\neg Y$
$(X \vee Y)$	X	Y
$(X \supset Y)$	$\neg X$	Y
$(X \subset Y)$	X	$\neg Y$
$(X \uparrow Y)$	$\neg X$	$\neg Y$
$\neg(X \downarrow Y)$	X	Y
$\neg(X \not\supset Y)$	$\neg X$	Y
$\neg(X \not\subset Y)$	X	$\neg Y$

prolog

Conjunctive Normal form

The algorithm that converts a propositional formula into CNF proceeds step by step. Basic transitions are:

replace $< \dots [\dots \beta \dots] \dots >$ by $< \dots [\dots \beta_1, \beta_2 \dots] \dots >$

replace $< \dots [\dots \alpha \dots] \dots >$ by $< \dots [\dots \alpha_1 \dots], [\dots \alpha_2 \dots] \dots >$

replace $< \dots [\dots \neg\neg\alpha \dots] \dots >$ by $< \dots [\dots \alpha \dots] \dots >$

prolog

Conjunctive Normal form

Un tel algorithme fonctionne, grâce au théorème du remplacement, par le remplacement de toutes les formules α et les formules β, selon les règles suivantes :

étape vers la forme normale conjonctive :

. remplacer <…[…β…]…> par <…[…β_1, β_2…]…>

. remplacer <…[…α…]…> par <…[…α_1…], […α_2…]…>

étape vers la forme normale disjonctive :

. remplacer […<…α…>…] par […<…α_1, α_2…>…]

. remplacer […< … β…>…] par […< …β_1…>, < …β_2…>…]

prolog

Write $((A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C)))$
in CNF.

prolog

Arbres de preuve (tableaux)

Un arbre représente la disjonction de ses branches.
Une branche représente une conjonction de formules propositionnelles.

Pour X une formule propositionnelle composée sur la branche B, la croissance de l'arbre consiste à :

- si X est de type α, rajouter α_1 puis α_2 en fin de B.
- si X est de type β, créer un nœud et deux nouvelles branches B_1, B_2 en fin de B, y rajouter β_1 et β_2 respectivement.

Une branche est fermée si X et $\neg$X y apparaissent.
Un arbre est fermé si toutes ses branches sont fermées.

Un arbre de preuve pour F est un arbre fermé croissant à partir de $\{\neg F\}$.

prolog

Proof by refutation

•Prolog works by refutation.

```
grand_parent(X,Y) :- parent(X,Z), parent(Z,Y).
[grand_parent(X,Y), ¬parent(X,Z), ¬parent(Z,Y)]
```

```
a :- b, c.
c :- e.
e :- b.
d.
b :- d.
```

```
[a, ¬b, ¬c]
[c, ¬e]
[¬b, e]
[d]
[¬d, b]
```

```
?- a.
True
```

prolog

Proof by resolution

- a *sequence* is a conjunction of lines
- a *line* is a generalized disjunction (clause)
- **growth** of the sequence:
- if a clause reads as [... β...], insert a new line: [[...$\beta 1$, $\beta 2$ ...]
- if a clause reads as [... α ...], insert two new lines: [... $\alpha 1$ ...] and [... $\alpha 2$ ...]
- add new lines by replacing $\neg\neg X$ by X, $\neg$T by $\perp$ and $\neg\perp$ by T
- **resolution**: from lines [... X ...] and [... $\neg X$ ...] create the line [... ... ...], i.e. concatenate the lines leaving aside all occurrences of X and of $\neg X$)
- a *proof of X* by resolution is a sequence including the [$\neg X$] line (*goal*) and containing an empty clause [ ].
- X is a tautology if and only if X has a proof by resolution.

prolog

Prove, using the resolution algorithm:
$((A \supset B) \wedge (B \supset C)) \supset \neg(\neg C \wedge A)$

a. [$\neg(((A \supset B) \wedge (B \supset C)) \supset \neg(\neg C \wedge A))$]
b. [$((A \supset B) \wedge (B \supset C))$] développement de a.
c. [$(\neg C \wedge A)$] développement de a.
d. [$(A \supset B)$] développement de b.
e. [$(B \supset C)$] développement de b.
f. [$\neg A$, B)] réécriture de d.
g. [($\neg B$, C] réécriture de e.
h. [$\neg C$] développement de c.
i. [A] développement de c.
j. [B] résolvante de f et i.
k. [C] résolvante de g et j.
l. [ ] résolvante de h et k.

prolog

Résolution et Prolog

```
parent(pam, bob).
parent(bob, tom).
gparent(X,Y) :-
     parent(X,Z),
     parent(Z,Y).
```

[parent(pam, bob)]
[parent(bob, tom)]
[gparent(X,Y), ¬ parent(X,Z), ¬parent(Z,Y)]

```
?- gparent(pam, tom).
```

[¬gparent(pam, tom)]

prolog

Déduction (syntaxique)

$S \mid\!- X$

X can be proven from S.

$\mid\!- X$

X can be proven.

Show that $S \cup \{X\} \mid\!- Y$ iff $S \mid\!- (X \supset Y)$

prolog

Show using deduction theorem that (*modus ponens*)

$\{P, (P \supset Q)\} \; |\!- \; Q$

$\{(P \supset Q)\} \; |\!- \; (P \supset Q)$	trivial
$\{P, (P \supset Q)\} \; |\!- \; Q$	(par th. déduction)

Montrer, en utilisant le théorème de déduction, que $((P \supset (Q \supset R)) \supset (Q \supset (P \supset R)))$ est un théorème.

$\{(P \supset (Q \supset R), P, Q\} \; |\!- R$ (par deux *modus ponens*)
$\{(P \supset (Q \supset R), Q\} \; |\!- \; (P \supset R)$ (par th. déduction)
$\{(P \supset (Q \supset R)\} \; |\!- \; (Q \supset (P \supset R))$ (par th. déduction)
$|\!- \; ((P \supset (Q \supset R) \supset (Q \supset (P \supset R)))$ (par th. déduction)

prolog

Axiomatic systems

Un système de Hilbert

schéma d'axiome 1 : $(X \supset (Y \supset X))$

schéma d'axiome 2 : $(X \supset (Y \supset Z)) \supset ((X \supset Y) \supset (X \supset Z))$

schéma d'axiome 3 : $(\perp \supset X)$

schéma d'axiome 4 : $(X \supset \mathrm{T})$

schéma d'axiome 5 : $(\neg\neg X \supset X)$

schéma d'axiome 6 : $(X \supset (\neg X \supset Y))$

schéma d'axiome 7 : $(\alpha \supset \alpha_1)$

schéma d'axiome 8 : $(\alpha \supset \alpha_2)$

schéma d'axiome 9 : $((\beta_1 \supset X) \supset ((\beta_2 \supset X) \supset (\beta \supset X)))$

règle d'inférence (*modus ponens*) : $\dfrac{X \quad (X \supset Y)}{Y}$

prolog

Montrer que $((\neg X \supset X) \supset X)$ est un théorème, pour toute formule X.

On part de l'axiome 9 avec $\beta = (\neg X \supset X)$:

$(\neg\neg X \supset X) \supset ((X \supset X) \supset ((\neg X \supset X) \supset X))$

Or $(\neg\neg X \supset X)$ est un axiome (sh. axiome 5).

Si l'on prouve $(X \supset X)$, alors on a le résultat par modus ponens.

Or $(X \supset X)$ résulte de la séquence suivante :

$(X \supset ((X \supset X) \supset X)) \supset ((X \supset (X \supset X)) \supset (X \supset X))$ (sh. axiome 2 et

$X \supset ((X \supset X) \supset X)$ sh. axiome 1 avec $Y = (X \supset X)$)

$(X \supset (X \supset X)) \supset (X \supset X)$ (par modus ponens)

$(X \supset (X \supset X))$ (sh. axiome 1)

$(X \supset X)$ (par modus ponens)

prolog

Soudness and completeness

Correction

Soient F une formule propositionnelle et S un ensemble de formules propositionnelles,

s'il existe une séquence dérivant de $S \cup \{\neg F\}$ et contenant une clause vide,

ou un arbre fermé croissant à partir de $S \cup \{\neg F\}$,

alors $S \models F$.

Complétude

Soient F une formule propositionnelle et S un ensemble de formules propositionnelles,

si $S \models F$, alors il existe une séquence dérivant de $S \cup \{\neg F\}$ et contenant une clause vide,

et un arbre fermé croissant à partir de $S \cup \{\neg F\}$.

prolog

Lemme d'Hintikka

- Un ensemble H de formules propositionnelles ayant les propriétés qui suivent s'appelle un ensemble d'Hintikka.

1. Il est cohérent :

   Pour tout A symbole propositionnel, soit $A \notin H$, soit $\neg A \notin H$.

   $\perp \notin H$; $\neg \top \notin H$

2. Il est clos :

   Si $\neg\neg X \in H$, alors $X \in H$

   Si $\alpha \in H$, alors $\alpha_1 \in H$ et $\alpha_2 \in H$.

   Si $\beta \in H$, alors $\beta_1 \in H$ ou $\beta_2 \in H$.

- Un ensemble d'Hintikka est satisfiable.

prolog

Completeness

- If one cannot derive the empty clause by resolution from S, then S can be augmented into an Hintikka set and is therefore satisfiable.

- If X is a tautology, then one can derive the empty clause from X; otherwise one could derive [ ] from $\{\neg X\}$ and it would be satisfiable.

prolog

Complétude (axiomatique)

- Tout ensemble de formules qui *ne peut pas* conduire par dérivation à une formule X quelconque donnée peut être complété en un ensemble d'Hintikka, et est donc satisfiable.

- Supposons maintenant que $S \models X$. Ceci entraîne que $S \cup \{\neg X\}$ n'est pas satisfiable.
  Donc $S \cup \{\neg X\}$ peut conduire par dérivation à n'importe quoi, en particulier à X.

- Grâce au théorème de déduction, $S \vdash_h (\neg X \supset X)$

- donc $S \vdash_h X$ puisque $((\neg X \supset X) \supset X)$ est un théorème.

prolog

Complexité (en temps ou espace)

- SAT (satisfiability of a propositional expression) is an NP-complete problem.

- SAT restricted to clauses de Horn is a P-complete problem.

prolog