Logique

Romann Szczepaniak

November 26, 2025

1 Un peu d'histore

On a besoin de formaliser le raisonemment : quelle idée découle de quoi.

Sauf qu'au XXe siècle, il y a eu une crise des fondements mathématiques.

Le paradoxe de Bertrand Russel : L'ensemble U des ensembles qui ne se contiennent pas eux-mêmes

Alors, U est-il dans U ? Si oui, il se contient lui-même, donc il ne devrait pas l'être. Sinon, il devrait se contenir, mais un paradoxe. Donc, on pourrait prouver tout et son contraire

Vient alors Hilbert. Il utilise un ensemble réduit de principes de raisonnements qui est suffisant et cohérent pour prouver toutes les mathématiques.

Mais Gödel vient dans l'histoire : Soit ce système d'axiomes est incomplet, soit il est incohérent.

Alan Turing s'inspire de certains aspects des théorèmes de Gödel pour imaginer les machines de Turing. Ses idées ont permis la création des premiers ordinateurs après la seconde guerre mondiale

1.1 Syntaxe, sémantique

Syntaxe	**Sémantique**
Formule	Modèles
Programme	Exécution, effet

Exemple 1 (Exemple d'applications de la logique). *Démonstration automatique de théorèmes*
Vérifications et optimisation de programmes
Certification de programmes (Logique de Hoare)

2 La logique propositionnelle

S'intéresse à l'articulation de la notion de vérité.

La logique propositionnelle se constitue de deux parties : la syntaxe et la sémantique

2.1 Les atomes

$\bot$: Absurde

$\top$: Tautologie

2.2 Les formules composites

Les unaires

$\neg$: La négation

Les binaires : $\wedge$: conjonction $\vee$: disjonction $\implies$: implication $\iff$: équivalence

2.3 Ecriture des formules

Ordre de priorité

$\neg$ puis $\wedge$ puis $\vee$ puis $\implies$ puis $\iff$

On peut écrire les formules sous forme d'arbre binaire

Var (ϕ) = nom des variables de la formule

La hauteur d'une fonction est le nombre d'opérations imbriqués

3 Sémantique

Chaque variable propositionnelle va être associée à un "énoncé en Français simple qui peut être vrai ou faux"

$p \wedge q$ est vrai $\iff$ p, q sont vrais $p \vee q$ est vrai $\iff$ p vrai ou q vrai $\neg p$ est vrai $\iff$ p faux

La valuation est une fonction ν qui associe l'ensemble $\{\text{vrai}, \text{faux}\}$ à l'ensemble $\{0, 1\}$ tel que $\nu(\text{vrai}) = 1$ et $\nu(\text{faux}) = 0$

Par exemple, $[\![\top, \nu]\!] = 1$

Une formule est satisfiable si il existe une valuation ν telle que la formule soit égale à 1

Deux formules ϕ et ξ sont équivalentes sémantiquement, noté $\phi \equiv \xi$ lorsque pour toute valuation ν, $[\phi, \nu] = [\xi, \nu]$

La relation $\equiv$ est une congruence sur les formules. C'est une relation d'équivalence

Elle est réflexive, symétrique et commutative

Voir le tableau des propriétés algébriques

Par exemple, la distributivité de $\wedge$ sur $\vee$ ou les lois de De Morgan

4 Formes normales

Notions

i. un litéral est toute formule de forme x ou $\neg x$ tel que x est une variable

ii. Une forme normale disjonctive : C'est une disjonction de conjonctions de litéraux

iii. Une forme normale disjonctive : C'est une conjonction de disjonctions de litéraux

Mise en forme sous FMD

- On transforme les $\implies$ et les $\iff$
- On pousse les négations au niveau des variables
- On applique la distributivité

5 Le problème SAT

Ce problème est de savoir si une formule normale conjonctive est satisfiable. Ce problème est NP-complet : On ne peut pas le résoudre en temps polynomial avec un algorithme non déterministe, et tel que tout problème NP peut se résoudre à un problème SAT

5.1 SAT comme langage de programmation

Premier problème

On dispose de m pigeons et de n nids. Le problème a une solution si

i. Chaque pigeon a un nid

ii. Chaque nid a au plus un pigeon

On introduit les variables $p_{i,j}$ pour $i \in \{1, m\})$, $j \in \{1, n\}$ interprète $p_{i,j}$ par le pigeon i est présent dans le nid j

Chaque pigeon doit se trouver dans (au moin) un nid :

$$(p_{1,1} \wedge p_{1,2} \wedge p_{1,3}) \vee (p_{2,1} \wedge p_{2,2} \vee p_{2,3})\,.$$

Maintenant, dans chaque nid, il y a au plus un pigeon

$$(\neg p_{1,1} \vee \neg p_{2,1}) \wedge (\neg p_{1,2} \vee \neg p_{2,2}) \wedge (\neg p_{1,3} \vee \neg p_{2,3})\,.$$

Maintenant, chaque pigeon doit se trouver dans au plus un nid (formule longue mais ok)

Notation

Les formules i et $\neg i$ sont des litéraux

les formules $(i_1 \vee i_2 \vee i_3 \cdots \vee i_n)$ sont des clauses

Exemple 2 (Colorier l'Australie). *On définit les variables :*

$C\,\{r, v, b\}$ *: Les couleurs qu'on utilisera*

$E = \{WA, NT, Q, SA, NSW, V, T\}$ *les états*

$L = \{(WA, NT) \ldots, (V, T)\}$ *l'ensemble des pays limitrophes*

La variable $x_{e,c}$ *décrit si l'état* e *est colorié avec la couleur* c

On peut voir la notation $\bigwedge_{e \in E}$ *: "Pour tout état e"*

On peut voir la notation $\bigvee_{e \in E}$ *: "Il existe (au moins) un état e"*

On a les contraintes suivantes :

i. Tout état a aun moins une couleur

$$\phi_1 = \bigwedge_{e \in E} \bigvee_{c \in C} x_{e,c}.$$

On peut

ii. Aucun état n'a deux couleurs

$$\phi_2 \bigwedge_{e \in E, (c_1, c_2) \in C | c_1 \neq c_2} \neg x_{e,c_1} \vee \neg x_{e,c_2}.$$

iii. Les états limitrophes n'ont pas la même couleur :

$$\phi_3 = \bigwedge_{(e_1,e_2) \in L} \bigvee_{c \in C} \neq x_{e_1,c} \vee \neq x_{e_2,c}.$$

WOOCLAP

5.2 Le format DIMACS

On va intéragir avec des solveurs SAT en python

Les solveurs SAT représentent les variables par des entiers strictement positifs : k représente la variable x_k

Le littéral $\neq x_k$ est représenté par $-k$

Nous passerons au solveur SAT un fichier texte dont voici la forme suivante :

Code Python

```
1
2    # Ce n'est pas du python
3
4   p cnf 5 2
5   1 -5 0
6   2 5 -1  0
```

Le 0 donne la fin de la clause

Voir le diaporama du cours pour voir la modélisation du problème des pigeons dans le format DIMACS (cf. Le format DIMACS : Le dilemme des pigeons)

Utiliser avec Python

Nous utiliserons python-sat qui permet d'installer et d'utiliser des solveurs SAT en python

On utilisera les fonctions encode(i, j) qui a un pigeon i et un nid j associe un nombre et decode(n) sa fonction inverse

On créera les clauses avec les listes en compréhension

Revoir les listes en compréhension si besoin

5.3 Le dilemme des pigeons en python

Code Python

```python
pigeons = [1, 2]
nids = [1, 2, 3]

# Au moins un pigeon par nid
[[encode(i, j) for j in nids] for i in pigeons]

# Au plus un pigeon par nid
[[-encode(i,j), -encode(k,j)] for i in pigeons for k in
   pigeons for j in nids if i<k]

...
```

On concatène les listes

Voir le diapo de cours pour voir l'exemple avec le coloriage de l'Australie

WOOCLAP

6 Transformer un problème de satisfiabilité en problème SAT

Les problèmes SAT sont des formules sous FNC Mais pour certaines formules, c'est pas possible de calculer la FNC (trop grand)

On utilisera la transformation de Tseitin pour obtenir un problème SAT :

De taille linéaire dans la taille de la formule de départ

Qui nous permette de trouver une valuation qui satisfasse cette formule

Voir l'algorithme d'un SAT solver

6.1 Transformation de Tseitin

Cette transformation est de taille linéaire par rapport à l'entrée, et nous permet de trouver une valuation qui satisfasse cette formule

Cette formule n'est pas équivalente, mais équisatisfiable

Comment faire ?

On fixe une formule ϕ dont nous voulons vérifier la satisfiabilité

i. On représente la formule sous forme d'arbre

   - Si cette formule est une variable, alors la variable associée est cette variable
   - Si cette sous formule n'est pas une variable, on lui associe une variable fraîche, qui n'est pas utilisée ailleurs

ii. On construit une sous formule grâce aux transformations de Tseitin (voir le tableau, sera donné à l'exam)

7 Résolution algorithmique SAT

Nous allons voir l'algorithme Davis-Putnam-Logemann-Loveland (algo DPLL)

Les opérations de cet algo sont :

i. Les simplifications

ii. La propagation unitaire

iii. L'élimination des littéraux purs

iv. La recherche de cas

L'algo va donner des valuations partielles

- Chaque règle assigne une valeur de vérité à une variable
- Les clauses sont simplifiées en fonction des valuations
- Si une recherche conduit à l'absurdité, on revient en arrière (backtracking)
- Les solveurs-SAT modernes sont basés sur cette méthode, mais il y a encore d'autres techniques

7.1 L'algorithme en détail

i. On élimine les variable superflues

ii. De même $\bar{I}$ est faux, on peut éliminer toute clause qui le contient

$$a \wedge (\neg a \vee b_1 \vee \cdots \vee b_n) \wedge (a \vee c \vee \cdots \vee c_n)\,.$$

$$\implies a \wedge (b_1 \vee \cdots \vee b_n) \wedge \cancel{(a \vee c \vee \cdots \vee c_n)}.$$

   Par contre, si on il y a $a \wedge \neg a$, l'algo s'arrête et il n'est pas satisfiable

iii. Elimination des litéraux purs

   Si on a que des $\neq a$, on prend $a = 0$ et on dégage tous les a

iv. Recherche par cas

- On choisit une variable pivot, disons a
- On choisit une valeur pour a
- On simplifie le problème en propageant la valeur choisie

Si on arrive à une contradiction, on revient en arrière et on change la valuation d'une variable

Si les deux cas donnent une contradiction, elle n'est pas satisfiable

L'ordre des opérations est primodial. La recherche par cas est un arbre.

Voir exemple sur le diapo.

(Il faut écrire les opérations que l'on fait avec un arbre)

8 Terme : Définition de la logique propositionnelle

8.1 Syntaxe des termes

On se fixe un ensemble infini de variables $F = \{x_1, x_2, \dots\}$ et un ensemble $\mathcal{F} = \{f, g, \dots\}$ de symboles de fonctions (appelées signature) avec l'application arité qui est le nombre de paramètres de la fonction

Exemple 3 (Exemple). $\mathcal{F} = \{\varnothing[0], \cup[2], \cap[2]\}$

Ensemble des termes sur $\mathcal{F}$ et X

Les termes sur $\mathcal{F}$ et X sont définis par induction comme :

- $x \in X$ est un terme
- $f \in \mathcal{F}, \text{arite}(f) = 0$ est un terme
- $f(t_1, \dots, t_n)$ est un terme si f est d'arité n et que les t_i sont des termes

Un terme est clos s'il ne contient aucune variable, et on note $\mathcal{T}(\mathcal{F})$ l'ensemble des termes clos

Exemple 4 (Exemple). *Si on considère* $\mathcal{F} = \{\varnothing[0], \cap[2], \cup[2]\}$ *et* $X = \{a, b\}$

- $\cup(a, b) \in \mathcal{T}(\mathcal{F}, X)$
- $\cap(\cup(\varnothing, a), \varnothing) \in \mathcal{T}(\mathcal{F}, X)$

8.2 Substitution

La substitution consiste à remplacer des variables par des termes :

Formellement, une substitution est une application d'un sous ensemble de X dans $\mathcal{T}(\mathcal{F}, X)$.

Une substitution est dite close si elle aboutit à un élément de $\mathcal{T}(\mathcal{F})$

Pour $t_1, \dots, t_k$ des termes de $\mathcal{T}(\mathcal{F}, X)$, on note $\{x_1/t_1, \dots, x_k/t_k\}$ la substitution ς telle que $\varsigma(x_i) = t_i$

Exemple 5. *Substitution*

Si on considère $\mathcal{F} = \{0[0], s[1], +[2], \times[2]\}$ *et* $X = \{a, b, c\}$

...

Soit t un terme et σ une substitution. On définit t_σ par induction sur t comme :

-

9 Sémantique

Comment interpréter les termes ?

On va définir une $\mathcal{F}-$ algèbre

Définition 1. *Une $\mathcal{F}$-algèbre est une structure $\mathcal{M}\left(D_\mathcal{M}, \left\{f^\mathcal{M}_{f\in\mathcal{F}}\right\}\right)$ où $D_\mathcal{M}$ est un ensemble non vide, appelé domaine, et pour chaque symbole de fonction $f \in \mathcal{F}_n$, on a :*

$$f^\mathcal{M} : D^n_\mathcal{M} \mapsto D_\mathcal{M}.$$

Voir les exemples dans le cours

Comment interpréter un terme qui n'est pas clos ?

Pour interpréter un terme, on se base sur :

Définition 2. *Affectation : une $\mathcal{M}$-affectation de X est une application $\lambda : X \mapsto D_\mathcal{M}$*

On note $[x_1 \mapsto a_1, \ldots, x_n \mapsto a_n]$ une $\mathcal{M}$-affectation tel que $\lambda(x_i) = a_i$

10 Problème d'unification

On s'intéresse à trouver une substitution qui rend égaux, deux à deux, des termes $t_1, \ldots, t_n$ et $u_1, \ldots u_n$

Définition 3 (Unifiable). *Un ensemble $E = \{t_1 = u_1, \ldots, t_n = u_n\}$ est unifiable s'il existe une substitution σ telle que pour tout $i \in \{1, \ldots, n\}$on a $t_{i\sigma} = u_{i\sigma}$*

Il peut y avoir plusieurs unificateurs, mais si l'ensemble des solutions n'est pas vide, un unificateur principal peut toujours être identifié

10.1 Unificateur

Il y a plusieurs algos, mais on va utiliser l'algorithme d'unification par réécriture, qui est juste un ensemble de règles mécaniques permettant de réécrire les éléments d'un ensemble d'équations constituant un problème d'unification.

Voir le tableau du cours

11 Vers la logique du premier ordre

11.1 Syntaxe

On se fixe un ensemble infini de variables X, un ensemble $\mathcal{F}$ de symboles de fonctions et un ensemble $\mathcal{R}$ de symboles de prédicats avec l'application arité qui donne l'arité pour chaque symbole de prédicats.

Exemple 6 (Exemple). *Je sais pas*

i. $X = \{x, y, z, w, t\}$

ii. $\mathcal{F} = \{s[1]\}$

iii. $\mathcal{R} = \{divise\ [2]\}$

Formule atomique

étant donné une signature $\mathcal{S} = (\mathcal{F}, \mathcal{R})$

Une formule atomique sur $\mathcal{S}$ est de la forme

Exemple 7 (Exemple). *Voici des exemple de formules atomiques avec* $\mathcal{F} = \{s[1], 0[0]\}, \mathcal{R} = (N[1], P[2])$

- $N(s(0))$

11.2 Langage et modèle

Définir un langage, c'est :

- fixer un ensemble fini d'opé

VOIR LE COURS :)