Partie 5 - Logique

Chapitre ? - Logique Propositionnelle

I - Syntaxe

Nos valeurs booléennes “vrai” et “faux” sont

- $\top$ top
- $\bot$ bot

Il existe différents ordres pour la logique

Ordre 0 : $\neg, \wedge, \vee, \longrightarrow, \longleftarrow$

Ordre 1 : $\forall, \exists$...

V ensemble de variables

On nommera selon cette convention :

Propositions : p, q, r, ...

Variables propositionnelles : x, y, z, ...

```
type formula = Top | Bot | Var of variable
  | Not of formula
  | And of formula * formula
  | Or of formula * formula
  ...
```

Type inductif $\equiv$ Ensemble inductif

$E_0 = \{\text{Top}, \text{Bot}\} \cup V$

$E_{n+1} = E_n \cup \{\varphi_1 \bowtie \varphi_2 \mid \varphi_1, \varphi_2 \in E_n, \bowtie \in \{\wedge, \vee, \longrightarrow, \longleftrightarrow\}\} \cup \{\neg\varphi \mid \varphi \in E_n\}$

Connecteurs logiques : $\neg, \wedge, \vee, \longrightarrow, \longleftrightarrow$

Arité : nombre d’arguments. $\neg$ unaire (arité 1). Les autres sont binaires.

Une formule logique peut être représentée par un arbre. Aux feuilles on a des variables ou $\top$ ou $\bot$. Aux noeuds internes on a des connecteurs logiques.

$\longrightarrow$ On reprend le vocabulaires des arbres

Exemple :

$\text{var}(\varphi)$ est définit inductivement.

$\text{var}(\top) = \text{var}(\bot) = \emptyset$

...

Syntaxiquement, $(x \wedge y) \wedge z \neq x \wedge (y \wedge z)$

II - Sémentique

Valuation : $v : V \longrightarrow \{0, 1\}$. Le programme interprète les formules dans $\{V, F\}$

Evaluation des formules : On prolonge les valuations “par morphisme” pour toutes les formules.

Soit v une valuation. On définit $\bar{v} : E \longrightarrow \{0, 1\}$.

$\bar{v}(\bot) = 0$

$\bar{v} = 1$

$\bar{v} = v(x)$

$\bar{v}(\neg\varphi) = 1 - \bar{v}(\varphi)$

$\bar{v}(\varphi \bowtie \psi) =$ table de $\bowtie$ appliquée à $\bar{v}(\varphi)$ et $\bar{v}(\psi)$

Page 7 du poly :

Notation : $v \vDash \varphi \Leftrightarrow v(\varphi) = 1$

avec une barre c'est $\Leftrightarrow v(\varphi) = 0$

$\bar{v}(\varphi) = [\![\varphi]\!]_v$

$v \vDash \varphi$ "est un model satisfait"

satisfaisable / satfisfiable

tautologique = valide

antilogie

$\varphi \vDash \psi \longleftrightarrow \forall v, v \vDash \varphi \longrightarrow v \vDash \psi$

Equivalence logique : $\varphi \vDash \psi$ et $\psi \vDash \varphi$ parfois notée $\equiv$

4 - Liens Syntaxe Sémantique

- La valeur de $\bar{v}$ ne dépend que de $v(x)$ pour $x \in \text{var}(\varphi)$

Preuve par induction structurelle :

Soit v et w tels que $v(x) = w(x) \forall x \in \text{var}(\varphi)$.

- Si $\varphi = \top$ alors $\bar{v}(\varphi) = 1 = \bar{w}(\varphi)$
- Si $\varphi = \bot$ ...
- Si $\varphi = x$ alors $x \in \text{var}(\varphi)$ donc par hypothèse $v(x) = w(x)$ donc $\bar{v}(x) = \bar{w}(x)$
- Si $\varphi = \neg\psi$ alors par induction structurelle : $\bar{v}(\psi) = \bar{w}(\psi)$. Donc $\bar{v}(\varphi) = 1 - \bar{v}(\psi) = 1 - \bar{w}(\psi) = \bar{w}(\varphi)$

Remarque : L'hypothèse d'induction s'applique car var(ψ) $\subseteq$ var(φ).

- Si $\varphi = \varphi_1 \bowtie \varphi_2$ alors par induction $\bar{v}(\varphi_1) = \bar{w}(\varphi_1) \wedge \bar{v}(\varphi_2) = \bar{w}(\varphi_2)$.En effet, var($\varphi_1$) $\subseteq$ var(φ) et var(φ_2) $\subseteq$ var(φ).

Donc v et w coïncident sur var(φ_1) et var(φ_2).

Donc $\bar{v}(\varphi) = \bar{w}(\varphi)$ puisque le même calcul sur $\bar{v}(\varphi_1)$ et $\bar{v}(\varphi_2)$ s'opère dans le tableau de vérité.

- $\varphi \vDash \psi$ ssi $\varphi \longrightarrow \psi$ est une tautologie

$\varphi \vDash \psi$ ssi $\forall v$ valuation $v \vDash \varphi$ implique $v \vDash \psi$ ssi $\bar{v}(\psi) = 1$ lorsque $\bar{v}(\varphi) = 1$ ssi $\bar{v}(\varphi \to \psi) = 1$ d'après le tableau de vérité.

- φ sat ssi $\exists v : v \vDash \varphi$ ssi $\exists v : \bar{v}(\neg\varphi) = 0$ ssi $\neg\varphi$ n'est pas valide

5 - Prouver qu'une formule est une tautologie

5.1 - Table de vérités

φ tautologie ssi $\forall v$ valuation $v \vDash \varphi$

$\longrightarrow$ BruteForce

Nombre ∞ de valuations mais $\bar{v}(\varphi)$ ne dépend que de $v(x_1), v(x_2)...v(x_n)$ où $\{x_1, ...x_n\} = \text{var}(\varphi)$.

$\varphi = x \wedge y \longleftrightarrow y \wedge x$

$\text{var}(\varphi) = \{x, y\}$

$v(x), v(y)$	$x \wedge y$	$y \wedge x$	phi
0, 0	0	0	1
0, 1	0	0	1
1, 0	0	0	1
1, 1	1	1	1

Est une tautologie évidente

Autre exemple : $\varphi = ((x \vee y) \wedge (\neg y \vee z)) \longrightarrow x \vee z$

On notera ψ l'intérieur du membre gauche de φ

$v(x), v(y), v(z)$	$x \vee y$	$\neg y \vee z$	ψ	$x \vee z$	φ
0, 0, 0	0	1	0	0	1
0, 0, 1	0	1	0	1	1
0, 1, 0	1	0	0	0	1
0, 1, 1	1	1	1	1	1
1, 0, 1	1	1	1	1	1
1, 1, 0	1	0	0	1	1
1, 1, 1	1	1	1	1	1

Il sagit donc d'une tautologie

Coût de la méthode : n variable $\Longrightarrow 2^n$ lignes

5.2 - Substitutions

Dans l'arbre, on remplace les feuilles avec x par des sous-arbres ψ.

5.4 - Réduction à SAT

φ autologie ssi $\neg\varphi$ non satisfiable

Le problème :

- Entrée : Une formule φ
- Question : Est-elle satisfiable ?

S'appelle SAT

7 - Systèmes de connecteurs Complets

$C = \{\neg, \vee, \wedge, \longrightarrow, \longleftrightarrow\}$

On sait réécrire les formules utilisant les connecteurs $\longrightarrow$ et $\longleftrightarrow$ en des formules logiquement équivalentes et qui n'utilisent plus ces connecteurs. Autrement dit, on aurait pu construire E en prenant les connecteurs $C' = \{\neg, \vee, \wedge\}$.

En utilisant de Morgan, on réécrit $p \wedge q \equiv \neg(\neg p \vee \neg q)$.

Puis $C'' = \{\neg, \vee\}$ et même $C''' = \{\bar{\wedge}\}$ avec

$\bar{\wedge}$	0	1
0	1	1
1	1	0

On a $\neg p \equiv p \mathbin{\bar{\wedge}} p$

Donc $p \wedge q \equiv \neg(p \mathbin{\bar{\wedge}} q)$ car $p \mathbin{\bar{\wedge}} q \equiv \neg(p \wedge q)$

8 - FNC / FND

On se place dans le système complet $\{\neg, \wedge, \vee\}$.

Littéraux : Variables à négation de variable

Clause : Disjonction de littéraux : $x \vee \neg y \vee z \vee t \vee \neg u$

Conjonction de clause : $(x \vee \neg y \vee z) \wedge (x \vee y \vee z) \wedge (\neg x \vee z)$

Anticlause : $x \wedge y \wedge \neg z$

FND : $(\ldots) \vee (\ldots) \vee (\ldots)$

Une forme normale serait un cas où $\varphi \equiv \psi$ ssi forme-normale(φ) = forme-normale(ψ)

Prop : Pour toute formule φ il existe ψ_1 et ψ_2 des formules équivalentes à φ telles que ψ_1 en FNC (ou l'inverse).

Connaissant un FNC de φ, il est "facile" de trouver une FND de $\neg\varphi$

$\varphi = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{p_i} l_{i,j}$

$\neg\varphi = \bigvee_{i=1}^{n} \neg \bigvee_{j=1}^{p_i} l_{i,j} = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{p_i} \neg l_{i,j}.$

Deux cas :

- Si $l_{i,j}$ est une variable alors $\neg l_{i,j}$ est un littéral
- Si $l_{i,j}$ est la négation d'une variable x alors $\neg l_{i,j} \equiv x$.

8.3 - Mise en Forme Normale

1) Via table de vérité

On remarque qu'une FND se déduit directement de la table de vérité : on remarque les lignes avec du 1.

Une anticlose est similaire à une valuation. Une FND s'obtient comme la liste des valuations qui satisfont φ.

Rappel : Pas unicité de la FND. Ici $\varphi = (x \wedge \neg y) \vee (y \wedge z) \vee (x \wedge z) \vee (y \wedge \neg y)$. Le dernier terme peut être retiré.

Remarque : On appelle FND (ou FNC) canonique une FND telle que chaque anticlause (resp. clause) qui contient exactement une fois chaque variable. Alors, il y a unicité de la FND canonique à l'ordre prêt.

2) Via réécriture

Il s'agit de faire un parcours d'arbre avec un pattern-matching afin de remplacer certaines formes de sous-formules.

Il faudra faire ce parcours tant qu'il y aura quelque chose à modifier. On arrête la boucle quand la formule n'est plus modifiée.

```
to_fnc(F):
  - remplacer les implications et équivalences comme dans la section 7
  - Tant que possible
      - Trouver une sous formule de F de la forme !(F1 || F2)
        et la remplacer dans F par (!F1 && !F2)
      - Idem avec une sous formule de la forme !(F1 && F2)
      - Idem avec une sous formule de la forme (F1 || (F2 && F3))
        par (F1 || F2) && (F1 || F3)
      - Idem avec (F1 && F2) || F3
```

A retenir : La complexité reste exponentielle dans le pire des cas, mais il y aura des cas dans lesquels "cela se passe mieux" tandis que la construction de la table de vérité était exponentielle dans tous les cas. Notamment, en pratique sur de petits exemples, ce sera plus rapide que la table de vérité.

Remarque : On va identifier un type de pire cas : $\varphi_n = \bigvee_{i=1}^{n}(x_i \wedge y_i)$. Appliquer la distributivité du $\vee$ sur le $\wedge$ va se générer en formule équivalente à φ_n en FNC, mais de taille exponentielle (voir 8.3.1). Donc notre algorithme de mise en FNC s'exécutea sur φ_n en temps $\Omega(2^n)$ au moins.

Remarque : Complexité de la mise en FNC/FND : On a 2 méthodes en temps exponentiel.

Proposition : Il n'existe pas d'algorithme de mise en forme normale qui soit de complexité $O(n^K)$ pour un entrant K.

Preuve : $\forall \psi$ en FNC : $\psi \equiv \varphi_n, |\psi| \geq 2^n$

Preuve de la terminaison

On définit q par induction structurelle sur les formules :
$\forall x \in V q(x) = 2$
$q(\varphi_1 \wedge \varphi_2) = q(\varphi_1) + q(\varphi_2) + 1$
$q(\varphi_1 \vee \varphi_2) = q(\varphi_1)q(\varphi_2)$
$q(\neg\varphi) = 4^{q(\varphi)}$

On montre que q est un variant de boucle pour la boucle while de l'algorithme.

On fait le calcul pour 1 des 4 règles.
$q(\neg(F_1 \wedge F_2)) = 4^{q(F_1)+q(F_2)+1}$
et
$q(\neg(\neg F_1 \vee \neg F_2)) = 4^{q(F_1)}4^{q(F_2)} = 4^{q(F_1)+q(F_2)} < q(F_3)$

Petit rappel : On note x_i, y_i, z_i les valeurs des variables au début du i^e tour de boucle. Comme q est un variant :

$q(x_0, y_0, z_0) > q(x_1, y_1, z_1) > q(x_2, y_2, z_2) > \ldots > q(x_d, y_d, z_d)$.

On veut majorer d (le nombre d'itération du while) : $d \leq q(x_0, y_0, z_0) + 1$

Ici, la mise en FNC de φ s'exécute en temps [au plus] $O(q(\varphi))$

C'est non satisfaisant car $q(\varphi)$ peut être de la forme $4^{\cdot^{\cdot^{\cdot^{4^n}}}} \mid n = |\varphi|$.

3) CNF Rapide
Hors-programme

Il existe un algorithme qui étant donné φ produit ψ en FNC en temps polynomial tel que :

$\exists v : v \vDash \varphi \Leftrightarrow \exists w : w \vDash \psi$

On dit que φ et ψ sont équisatisfiables.

9 - Logique = Langage de Spécification

Problème du Pavage

n et m les dimensions de rectangles à carreler

$S = \{s_0, ..., s_{p-1}\}$ l'ensemble des tuiles

$s_i = (n_i, e_i, s_i, o_i)$

Ensemble des variables : $p_{i,j,k}$ vraie si la tuile s_k se trouve en position (i, j).

A chaque emplacement (i, j), une seule tuile

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{m} \bigwedge_{k=0}^{p-1} \left(p_{i,j,k} \rightarrow \bigwedge_{\substack{l=0 \\ l \neq k}} \neg p_{i,j,l} \right)$$

Pour chaque emplacement il y a une tuile dessus

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{m} \left(\bigvee_{k=0}^{p-1} p_{i,j,k} \right)$$

10 - Problème SAT

C'est le nom que l'on donne au problème suivant :

Entrée : φ

Question : φ est-elle satisfiable ?

Remarque : φ SAT $\Leftrightarrow \neg\varphi$ n'est pas valide. Le problème SAT est "équivalent" à la question "est-ce que φ est une tautologie".

1) BruteForce==== 1) BruteForce

On a une fonction d'évil renvoie uation `eval v phi` renvoie $\bar{v}(\varphi)$.

On énumère les évaluations jusqu'à en trouver une qui satisfasse φ.

Correspond à l'approche "table de vérité"

2) Algorithme de Quine

Algorithme permettant de savoir si φ est satisfiable

```
is_sat(phi):
  if var(phi) = void:
    renvoyer True ou False # phi vraie ou fausse
  else:
    for x in var(phi):
      if is_sat(phi[x <- Top])
        return True
      else:
        return is_sat(phi[x <- Bot])
```

3) Raffinement dans le cas d'une CNF

$\longrightarrow$ Le vrai algorithme de Quine

Littéral : x ou $\neg x$

```
type litteral =
  | Lit of variable
  | Nlit of variable
```

Clause : Disjonction de littéraux

```
type clause = litteral list
```

Remarque : La clause vide équivaut à $\bot$

FNC : Conjonction de clauses

```
type fnc = clause list
```

Remarque : La fonction vide équivaut à $\top$

```
# Précondition : phi en CNF
is_sat(phi):
  if phi est la fnc vide: return Tru
  if phi contient une clasuse vide: return False
  else:
    for x in var(phi):
      # On effectue un max de simplifications lors de la substitution
      if is_sat(phi[x<-Top]): return True
      else: return is_sat(phi[x<-Bot])
```

Mais que nous apporte la FNC ? Elle facilite les simplifications.

$\varphi[x \leftarrow \top] \longrightarrow$ pour chaque clause c
- Si x est dans c, on supprime la clause
- Si $\neg x$ est dans c, on retire $\neg x$ de la clause

Avec $\varphi[x \leftarrow \bot]$ analogue

Optimisation : Propagation des clauses unitaires : Si une clause c n'a qu'un seul littéral, on le met à directement à vrai.

Remarque : La CNF permet de rendre plus efficace les simplifications dans l'algorithme de BackTracking de Quine. Mais la mise en CNF est exponentielle, d'où l'intérêt de la CNF rapide.

Amicalement Grégoire