Université Paris 7 - Denis Diderot

Licence Sciences et Applications

Mention : Mathématiques, Informatique

IF1 - Introduction à l'informatique et la programmation

2008-2009

Amphi - 4-ème partie

Calcul booléen

Algèbre de Boole

- Introduite par Georges Boole au XIX^e siècle comme base mathématique du raisonnement logique

- calcul propositionnel : manipulation d'expressions conduisant à une valeur de vérité Faux ou Vrai.

- Vu du point de vue de la programmation en Java par exemple, il s'agit d'évaluer des expressions manipulant des valeurs de type boolean par exemple pour réaliser des tests dans des instructions conditionnelles

- Liens avec la théorie des ensembles telle que vue en cours de Maths

- le calcul de circuits : réalisation de circuits qui à partir de valeurs d'entrée binaires fournissent une valeur de sortie.

Des circuits simples sont réalisables par combinaison de transistors (élément de base permettant de réaliser un circuit tout ou rien, fonctionnant donc comme un interrupteur)

Le calcul propositionnel

- **nous nous intéresserons ici à l'aspect de ce calcul définissant des règles de calcul dans un domaine à deux valeurs $B = \{Faux , Vrai\}$, abrégées en F et V**

- **une *proposition* est une entité faisant référence à des événements ou états de choses. Ainsi "il y a du soleil" ou "l'avion est en retard" sont des propositions concrètes susceptibles d'être vraies ou fausses. Il en est de même dans un programme de l'expression $x + y < z$**

- ***les connecteurs logiques*.**

  Un connecteur $n-aire$ associe à n valeurs du domaine B (c'est-à-dire un élément de B^n) une valeur de B : pour une valeur n donnée, il en existe 2^{2^n}

  il existe 4 $connecteurs\ unaires\ \alpha_i$

proposition	α_1	α_2	α_3	α_4
F	F	F	V	V
V	F	V	F	V

  dont les $"petits\ noms"$ **sont :**

  α_1 **: la contradiction**

il existe 16 *connecteurs binaires* β_i

p_1	p_2	β_1	β_2	β_3	β_4	β	β_6	β_7	β_8
F	F	F	F	F	F	F	F	F	F
F	V	F	F	F	F	V	V	V	V
V	F	F	F	V	V	F	F	V	V
V	V	F	V	F	V	F	V	F	V

p_1	p_2	β_9	β_{10}	β_{11}	β_{12}	β_{13}	β_{14}	β_1	β_{16}
F	F	V	V	V	V	V	V	V	V
F	V	F	F	F	F	V	V	V	V
V	F	F	F	V	V	F	F	V	V
V	V	F	V	F	V	F	V	F	V

On y re retrouve des opérateurs de

- contradiction (β_1)
- tautologie (β_{16})

Opérateurs principalement utilisés :

- la conjonction (β_2), aussi appelée ET ou AND.
  La notation mathématique de ce connecteur est $\wedge$ et en Java on le note && dans sa forme paresseuse et & dans sa forme non paresseuse.
- la disjonction (β_8), aussi appelée OU ou OR.
  La notation mathématique de ce connecteur est $\vee$ et en Java on le note || dans sa forme paresseuse et | dans sa forme non paresseuse.
- l'équivalence (β_{10}) notée $\equiv$ en math
- l'implication (β_{14}) notée $\rightarrow$ en math

On peut y ajouter :

- β_1 , **négation de** β_2**, appelée incompatibilité ou NAND**
- β_9**, négation de** β_8**, appelée NOR**
- β_7**, négation de** β_{10}**, appelée XOR ou OU exclusif, notée ^ en Java**

Les expressions logiques ou booléennes

De même qu'on écrit des expressions arithmétiques avec des constantes numériques, ou des variables numériques et des opérateurs arithmétiques, on peut écrire des expressions logiques avec les constantes ou des variables propositionnelles et des connecteurs, d'où des règles de syntaxe

Exemples d'expressions

Les expressions suivantes utilisent :

- **des variables a, b et c**
- **les connecteurs ¬, □, □, ≡, □ et ˆ (pour XOR)**
- **les parenthèses pour lever toute ambiguïté**

$E_1 : ((a \square (\neg b)) \square (\neg(b \equiv c))) \square (a \hat{} c)$

$E_2 : (((\neg a) \square b) \square c)$

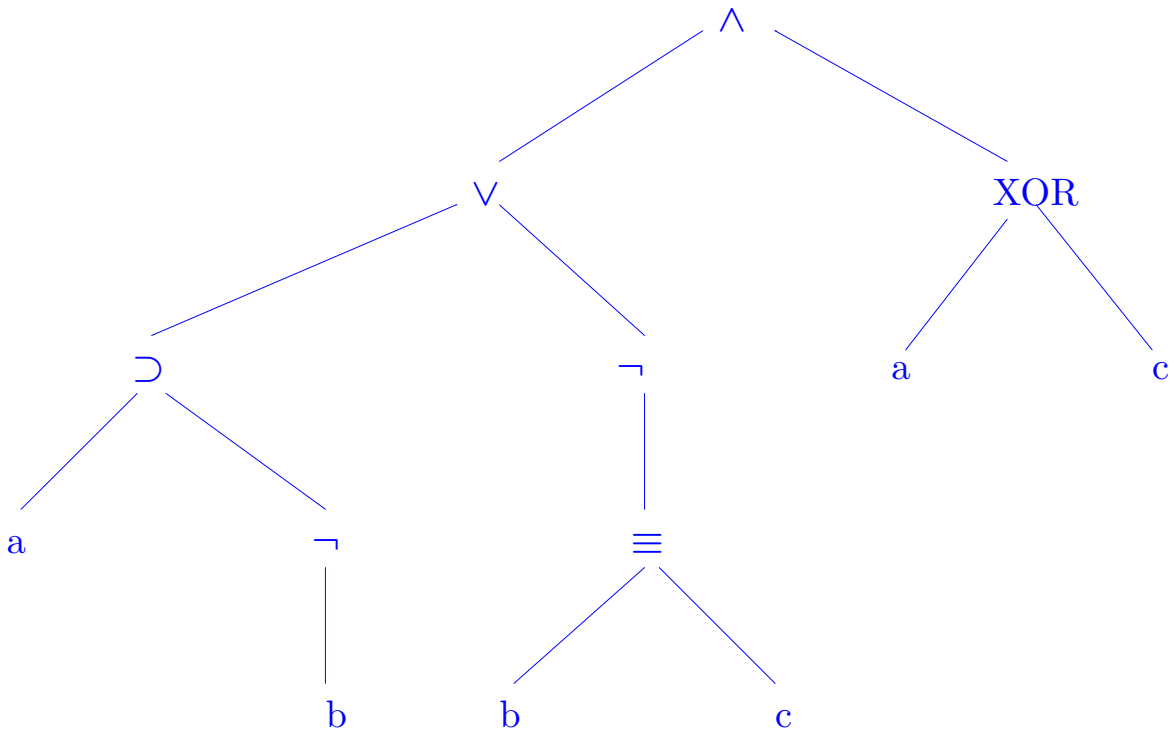
$E_3 : ((a \equiv b) \square ((\neg a) \square b))$

Remarque **: les priorités entre opérateurs allègent l'écriture. C'est typiquement le cas dans les langages de programmation : la négation est plus prioritaire que la conjonction, elle-même plus prioritaire que la disjonction**

Arbre associé à une expression booléenne

Comme c'est le cas pour les expressions arithmétiques, il est commode d'associer à une expression logique un arbre qui reflète la manière dont elle a été construite.

Pour l'expression E_1 considérée :



Formes dites polonaises des expressions

- la représentation classique des expressions (arithmétiques ou booléennes) suppose l'usage de parenthèses et/ou de priorités des opérateurs pour lever les ambiguïtés
- l'arbre syntaxique associé à une expression en reflète la construction
- il existe deux représentations linéaires des expressions (arithmétiques ou booléennes) associées à une expression et facilement déductible de l'arbre associé :

- **dans la forme polonaise préfixée, l'opérateur précède les opérandes.**

  Ainsi, a b sera représentée par b a.

- **dans la forme polonaise suffixée ou postfixée, l'opérateur suit les opérandes.**

  Ainsi, sous forme suffixe, a □ b sera représentée par a b □.

  La forme suffixée de l'expression E_1 est :

  a b ¬ □ b c ≡ ¬ □ a c ˆ □

Remarque sur la construction de ces représentations :

elle peut être réalisée par "une promenade" dans l'arbre associée vu comme un mur situé à sa gauche.

On écrit les caractères rencontrés en certaines circonstances :

- **pour obtenir la représentation préfixée, on note un caractère la première fois qu'on passe par la position où il se trouve (passage à l'ouest)**
- **pour obtenir la représentation postfixée, on note un caractère la dernière fois qu'on passe par la position où il se trouve (passage à l'est)**

Ce type de représentation peut également être utilisée pour les expressions arithmétiques.

Ainsi pour l'expression arithmétique vue en cours/TD (page 42)

un+cinq/deux+cinq*deux/trois/deux-un+deux-trois

on obtient

- **la forme préfixe :**

  - + - + + un / cinq deux / / * cinq deux trois deux un deux trois

- **la forme suffixe :**

  un cinq deux / + cinq deux * trois / deux / + un - deux + trois -

Quels connecteurs choisir ?

- **tous les connecteurs peuvent se déduire d'un nombre limité des autres**
- **différents choix sont possibles pour ces connecteurs primitifs et la simplicité des expressions dépend beaucoup de ce choix**

- **dans les langages de programmation on dispose généralement de la négation, la conjonction et la disjonction, voire du XOR comme en C ou Java**
  Ce choix est adapté aux types d'expression qu'on souhaite y décrire.

Il est redondant car de fait la négation et la conjonction, par exemple, suffisent :

- $a \square b$ **est équivalente à** $\neg((\neg a) \square (\neg b))$ **(c'est une des** $lois\ de\ Morgan$**)**
- $a\ XOR\ b$ **est par exemple équivalente à** $(a \square b) \square (\neg(a \square b))$

Tables de vérité

Il est d'usage d'associer à une expression de n variables une table donnant de manière exhaustive (c'est-à-dire pour les 2^n configurations possibles des valeurs des variables) la valeur de l'expression :
une telle table est appelée table de vérité.

Ainsi pour l'expression E_1, nous obtenons :

a	b	c	$\neg b$	α : $a \supset \neg b$	β : $b \equiv c$	γ $\neg\beta$	δ : $\alpha \vee \gamma$	ϵ : $a \barwedge c$	E : $\delta \wedge \epsilon$
F	F	F	V	V	V	F	V	F	F
F	F	V	V	V	F	V	V	V	V
F	V	F	F	V	F	V	V	F	F
F	V	V	F	V	V	F	V	V	V
V	F	F	V	V	V	F	V	V	V
V	F	V	V	V	F	V	V	F	F
V	V	F	F	F	F	V	V	V	V
V	V	V	F	F	V	F	F	F	F

Pour l'expression E_2 :

a	b	c	$(\neg a)$	$((\neg a) \vee b$	$E_1 = (((\neg a) \vee b) \supset c)$
F	F	F	V	V	F
F	F	V	V	V	V
F	V	F	V	V	F
F	V	V	V	V	V
V	F	F	F	F	V
V	F	V	F	F	V
V	V	F	F	V	F
V	V	V	F	V	V

Pour l'expression E_3 :

a	b	$(a \equiv b)$	$(\neg a)$	$(\neg a) \wedge b$	$((a \equiv b) \supset ((\neg a) \wedge b))$
F	F	V	V	F	F
F	V	F	V	V	V
V	F	F	F	F	V
V	V	V	F	F	F

Il est ainsi possible par exemple de

tester si deux expressions E et F sont ou non équivalentes

ce qui du point de vue logique s'exprime par le fait que l'expression

$E \equiv F$ est une tautologie

(c'est-à-dire est vraie quelles que soient les valeurs des variables la constituant)

On peut montrer par exemple que $\neg((\neg a) \square (\neg b))$ et $a \square b$ sont équivalentes :

a	b	$a \vee b$	$\neg a$	$\neg b$	$(\neg a) \wedge (\neg b)$	$\neg((\neg a) \wedge (\neg b))$
F	F	F	T	T	T	F
F	V	V	T	F	F	V
V	F	V	F	V	F	V
V	V	V	F	F	F	V

Forme normale disjonctive

Considérons une expression logique E de n variables p_1, p_2, . . . p_n.

- **pour chaque $i = 1, \ldots n$, désignons symboliquement par π_i l'une quelconque des expressions p_i ou $\neg p_i$**
- **la forme normale disjonctive E_D de l'expression E est l'expression qui lui est équivalente et est, pour un certain entier m de la forme**

$$\bigvee_{i=1}^{m} \left(\bigwedge_{j=1}^{n} \pi_j \right)$$

On la construit à partir des valeurs des variables pour lesquelles l'expression est vraie. Pour l'expression E_1 :

$$(\neg a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge c) \vee (a \wedge \neg b \wedge \neg c) \vee (a \wedge b \wedge \neg c)$$

ou encore sous une forme abrégée ($\bar{a}$ à la place de $\neg a$ et opérateurs $\square$ omis) :

$$\bar{a}\bar{b}c \square \bar{a}bc \square a\bar{b}\bar{c} \square ab\bar{c}$$

Problèmes de quantification

De nombreuses propositions ou propriétés, en particulier des programmes, s'expriment par des phrases telles que :

- **pour tout x, tout y et tout z entiers, si la propriété P_1 relie x et y (ce qu'on note $P_1(x, y)$, par exemple $x < y$) et si la propriété P_2 est vraie pour z (notée $P_2(z)$, par exemple $z > 0$), alors la propriété P_3 relie x, y et z (notée $P_3(x, y, z)$, comme par exemple $(x + z) < (y + z)$).**
  On note de manière symbolique une telle proposition :
  $$\forall x \forall y \forall z((P_1(x, y) \wedge P_2(z)) \Rightarrow P_3(x, y, z))$$
  et $\forall$ est appelé quantificateur universel
  Pour l'interprétation de P_1, P_2 et P_3 donnée, cette proposition est évidemment vraie. Pour d'autres elle est susceptible d'être fausse.

- **il existe x, tel que si la propriété P est vraie pour x ($P(x)$ comme par exemple "x est impair"), alors la propriété Q est également vraie pour x ($Q(x)$ comme par exemple "$x+1$ est une puissance de 2"). Une telle proposition s'exprime par :**

  $$\exists x(P(x) \Rightarrow Q(x))$$

  et $\exists$ est appelé quantificateur existentiel Pour l'interprétation de P et Q donnée, cette proposition est évidemment vraie. Pour d'autres elle est par contre susceptible d'être fausse.
- **des propositions plus compliquées peuvent être construites qui mélangent les différents types de quantifications, comme dans la suivante :**

  $$\exists x \exists y \forall z((A(x,y) \Rightarrow ((B(x,z) \Rightarrow B(z,y))$$

Passage à la négation

La principale difficulté de ce type de propositions est le passage à la négation

Disons simplement que :

- $\neg(\square x P(x))$ **est la proposition** $\square x(\neg P(x))$
- $\neg(\square x P(x)$ **est la proposition** $\square x(\neg P(x))$