

Université Paris 7 - Denis Diderot
 DEUG MASS-MIAS
 IF121 - Informatique fondamentale
 Calcul booléen

1

Algèbre de Boole

- Introduite par Georges Boole au XIX^e siècle comme base mathématique du raisonnement logique
- Du point de vue informatique, différents champs d'application ou points de vue :
 - le 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

2

- 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)

3

Le calcul propositionnel

- nous nous intéresserons ici à l'aspect de ce calcul définissant des règles de calcul dans un domaine à deux valeurs $\mathcal{B} = \{\text{Faux}, \text{Vrai}\}$, 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 \mathcal{B} (c'est-à-dire un élément de \mathcal{B}^n) une valeur de \mathcal{B} : pour une valeur *n* donnée, il en existe 2^{2^n}

4

◇ il existe 4 connecteurs unaires α_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** (toujours fausse)

α_2 : l'**identité** ou **affirmation**

α_3 : la **négation** notée \neg en math
et **!** en Java

α_4 : la **tautologie** (toujours vraie)

◇ il existe 16 connecteurs binaires β_i

p_1	p_2	β_1	β_2	β_3	β_4	β_5	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_{15}	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

5

On retrouve des opérateurs de

- **contradiction** (β_1)

- **tautologie** (β_{16})

Les opérateurs principalement utilisés sont:

- 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.

6

- l'**équivalence** (β_{10}) notée \equiv en math

- l'**implication** (β_{14}) notée \supset en math

On peut également distinguer:

- β_{15} , **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 \wedge 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 syntaxes

7

Exemples d'expressions

Les expressions suivantes utilisent:

◇ des variables a , b et c

◇ les connecteurs \neg , \vee , \wedge , \equiv , \supset et \wedge (pour XOR)

◇ les parenthèses pour lever toute ambiguïté

$$E_1 : ((a \supset (\neg b)) \vee (\neg(b \equiv c))) \wedge (a \wedge c)$$

$$E_2 : (((\neg a) \vee b) \supset c)$$

$$E_3 : ((a \equiv b) \supset ((\neg a) \wedge b))$$

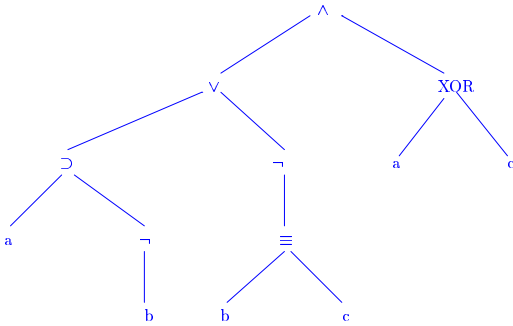
Remarque: les priorités entre opérateurs allègent l'écriture. C'est typiquement le cas dans les langages de programmation: la négation est plus prioritaire que la conjonction, elle-même plus prioritaire que la disjonction

8

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 :



9

Formes dites polonaises des expressions

- la représentation classique des expressions (arithmétiques ou booléennes) supposent 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 opérands. Ainsi, $a \wedge b$ sera représentée par $\wedge b a$. Si a et b sont des expressions, elles seront remplacées par leur équivalent sous la forme polonaise préfixée. La forme préfixée de l'expression E_1 est:

$$\wedge \vee \supset a \neg b \neg \equiv b c \wedge a c$$

10

- ◊ dans la forme polonaise suffixée ou postfixée, l'opérateur suit les opérands. Ainsi, $a \wedge b$ sera représentée par $a b \wedge$ sous forme suffixe. La forme suffixée de l'expression E_1 est:

$$a b \neg \supset b c \equiv \neg \vee a c \wedge \wedge$$

Remarque sur la construction de ces représentations :

elle consiste en "une promenade" le long d'un mur situé à sa gauche (parcours d'arbre) durant laquelle on écrit les caractères rencontrés en certaines circonstances:

- pour obtenir la représentation représentée préfixée, on note un caractère la première fois qu'on passe par la position où il se trouve
- pour obtenir la représentation représentée postfixée, on note un caractère la dernière fois qu'on passe par la position où il se trouve

11

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

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

$$\text{un} + \text{cinq} / \text{deux} + \text{cinq} * \text{deux} / \text{trois} / \text{deux} - \text{un} + \text{deux} - \text{trois}$$

on obtient

- la forme préfixe :

$$- + - + + \text{un} / \text{cinq} \text{deux} / / * \text{cinq} \text{deux} \text{trois} \text{deux} \text{un} \text{deux} \text{trois}$$

- la forme suffixe :

$$\text{un} \text{cinq} \text{deux} / + \text{cinq} \text{deux} * \text{trois} / \text{deux} / + \text{un} - \text{deux} + \text{trois} -$$

12

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 en large part de ce choix
- dans les langages de programmation on dispose généralement de la négation, la conjonction et la disjonction, voire comme en Java du XOR.

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:

- $a \vee b$ est équivalente à $\neg((\neg a) \wedge (\neg b))$ (c'est une des lois de Morgan)
- $a \text{ XOR } b$ est par exemple équivalente à $(a \vee b) \wedge (\neg(a \wedge b))$

13

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	$\neg b$	$\alpha: a \supset \neg b$	$\beta: b \equiv c$	$\gamma: \neg \beta$	$\delta: \alpha \vee \gamma$	$\epsilon: a \wedge c$	$E: \delta \wedge \epsilon$
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

14

Pour l'expression E_2 :

a	b	c	$(\neg a)$	$((\neg a) \vee b)$	$E_2 = (((\neg a) \vee b) \supset c)$
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

15

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

$\neg((\neg a) \wedge (\neg b))$ et $a \vee b$ sont équivalentes:

a	b	$a \vee b$	$\neg a$	$\neg b$	$(\neg a) \wedge (\neg b)$	$\neg((\neg a) \wedge (\neg b))$
F	F	<u>F</u>	T	T	T	<u>F</u>
F	V	<u>V</u>	T	F	F	<u>V</u>
V	F	<u>V</u>	F	V	F	<u>V</u>
V	V	<u>V</u>	F	F	F	<u>V</u>

16

Forme normale disjonctive

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

- pour chaque $i = 1, \dots, n$, 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 :

$$\underline{(\neg a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge c) \vee (a \wedge \neg b \wedge \neg c) \vee (a \wedge b \wedge \neg c)}$$

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

$$\underline{\bar{a}\bar{b}c \vee \bar{a}bc \vee a\bar{b}\bar{c} \vee abc}$$

17

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:

$$\forall x \forall y \forall z ((P_1(x, y) \wedge P_2(z)) \supset P_3(x, y, z))$$

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

18

" 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:

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

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:

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

Passage à la négation

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

Disons simplement que:

- $\neg(\forall x P(x))$ est la proposition $\exists x(\neg P(x))$
- $\neg(\exists x P(x))$ est la proposition $\forall x(\neg P(x))$

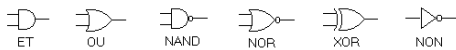
19

Point de vue électronique

- un calculateur numérique est constitué de circuits électroniques qui utilisent typiquement deux niveaux de tension *haut* et *bas* permettant la représentation de valeurs binaires (symboliquement désignés par 0 et 1).
- le calcul propositionnel permet la conception de tels circuits.
- des circuits élémentaires réalisant physiquement des opérations booléennes simples (négation, conjonction, disjonction, NAND ou NOR) sont faciles à construire dans différentes technologies (NAND et NOR sont utilisés dans les circuits intégrés): ces circuits élémentaires sont traditionnellement appelés **portes**

20

Notation des portes élémentaires

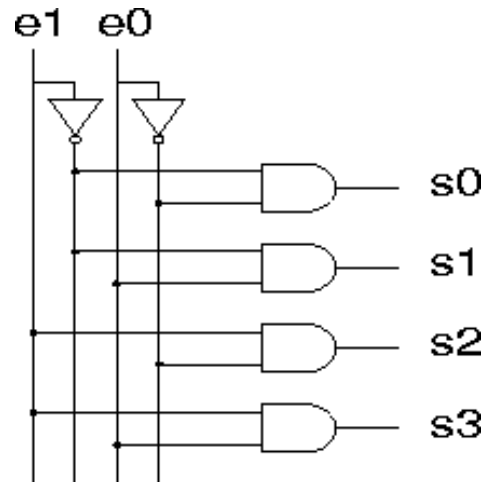


Par combinaison de ces circuits de base, on construit des circuits plus complexes:

- des circuits combinatoires: la valeur récupérée en sortie est exprimable comme une expression logique des variables d'entrée. Dans un circuit "idéal", le temps de propagation dans le circuit est nul, alors que dans la réalité le passage d'un état à l'autre ne l'est pas.

Des exemples de circuits de ce type sont les décodeurs et les multiplexeurs:

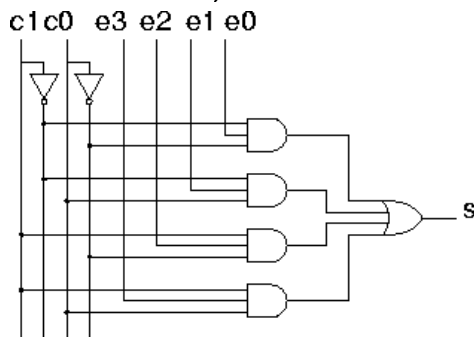
- ◇ les décodeurs (tels que les panneaux LED) permettent l'envoi d'une valeur à une sortie (n lignes d'entrée et 2^n lignes de sortie)



21

22

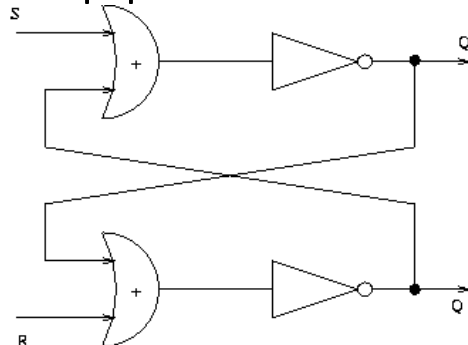
- ◇ les multiplexeurs qui fournissent en sortie l'une des valeurs d'entrée (les autres étant éliminées):



23

- des circuits séquentiels ou de mémorisation: la valeur de sortie dépend
 - ◇ de la valeur (état) des variables d'entrée
 - ◇ de la valeur (état) antérieur de certaines variables de sortie sous la forme d'un état interne

Un exemple d'un tel circuit est la bascule qui permet la mémorisation d'un bit:



24