C C

Le calcul propositionnel

Introduction

- ' $\mathcal{B}$
  { , } { , } { , } :
- *i i* .

A

- '

).

variables propositionnelles) ' .

'

(x + y) < z

, .

, b , ' :

b = (x + y) < z;

' , ,

' :

- : '
- 12 '
- 38 : '
- 41 : '
- 23- 4 : ' , ...

Les connecteurs logiques

- **Question préalable : combien existe-t-il d'applications (fonctions partout définies) différentes d'un ensemble fini de *m* éléments dans un ensemble fini de *n* éléments.**
- C , , **connecteurs logiques**
- , - ' -
  , ' - - $\mathcal{B}$, $\mathcal{B}$.
  , 2^{2^n} - .
- 2 **connecteurs -aires** : '
- 4 **connecteurs unaires**

A				

- :¬
- C, C++ : !

.

- 16 **connecteurs binaires**

A	B	A_1B	A_2B	A_3B	A_4B	A_5B	A_6B	A_7B	A_8B	A_9B	$A_{10}B$	$A_{11}B$	$A_{12}B$	$A_{13}B$	$A_{14}B$	$A_{15}B$	$A_{16}B$

	)		C / A A / C++	)	
1					
2	A	$A \wedge B$ $A \cdot B$			
7		$A \oplus B$			
8	))	$A \vee B$ $A + B$			
9	-	$A \overline{\vee} B$ $A \quad B$			:

\

10		$A \quad B$ $A \quad B$			
14		$A \supset B$ $A \quad B$			
15	-	$A \overline{\wedge} B$ $A \quad B$			A : A
16					

A

- , ' - -

{F,V} : 256.

, ' :

si x alors y sinon z

, x, y z, y z x

.

:

			, ,)

Aspects syntaxiques

- ' ' '

Les expressions ou formules logiques

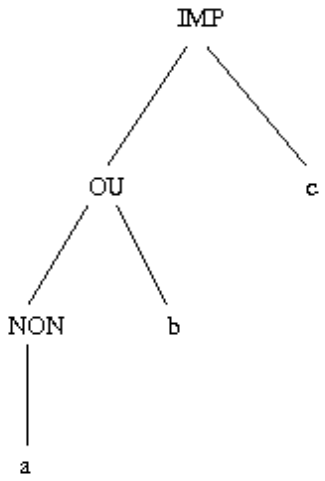
- ' $\mathcal{F}$
  :
  - $\mathcal{B} \subset \mathcal{F}$:
  - $\mathcal{V} \subset \mathcal{F}$:
  - E ,
  - E_1 E_2 ,
- ' ' $\mathcal{V}$
  a, b c :

  0 : ∧ ∨ ∧ ∨
-). C ' ' ,
  ' ,
  ').
- . C ' ' 0
- ' :
  - $E_1 : ((a \supset (\neg b)) \vee (\neg(\equiv c))) \wedge (a \hat{} c)$)
  - $E_2 : (((\neg a) \vee b) \supset c)$
  - $E_3 : ((a \equiv b) \supset ((\neg a) \wedge b))$
  - 4 : (a . b) + (c . (⊕ b)))
  - 5 : (¬ a . b) + (c . ¬ (a ⊕ b))

Arbre associé à une expression logique

C ' , '
.

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



Question : construire l'arbre associé à l'expression E_0 :

Question : construire l'arbre associé à l'expression E_4 : ⊕

Question : construire l'arbre associé à l'expression E_5 : ⊕

Question : construire l'arbre associé à l'expression $E_1 : ((a \supset (\neg b)) \vee (\neg(b \equiv c))) \wedge (a \barwedge c)$

Formes polonaises des expressions

-) ' /
- '
-)

' :

  - **dans la forme polonaise préfixe préfixée**), ' .
    A , *a b* , ***a*** ∧ ***b*** ∧ ***a b***.
    a b , -
    .

    , ' E_1 :

    ∧∨ ⊃*a*¬*b*¬≡*bc*⊼*ac*

  - **forme polonaise suffixe suffixée postfixée**), ' .
    A , *a b* , ***a*** ∧ ***b*** ***a b*** ∧
    a b , -
    .

    , ' E_1 :

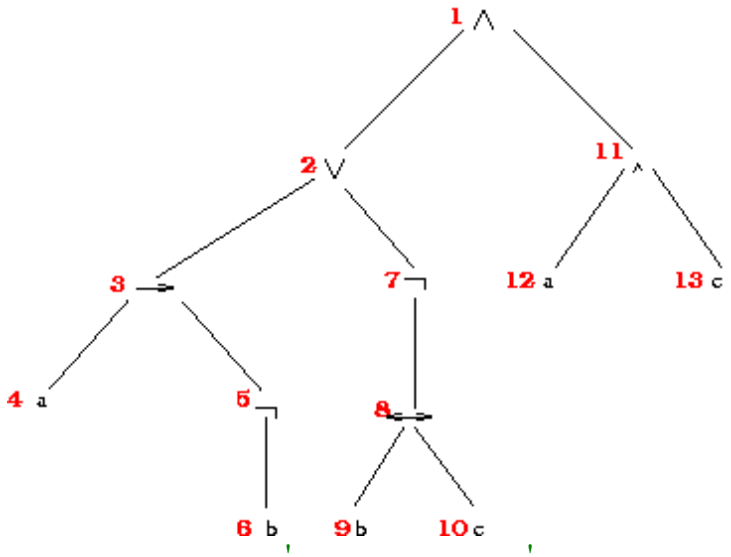
- **Construction des formes polonaises à partir de l'arbre d'une expression**

' ' ' - '), , , . C' ' **parcours d'arbre**. ' , ' .

:

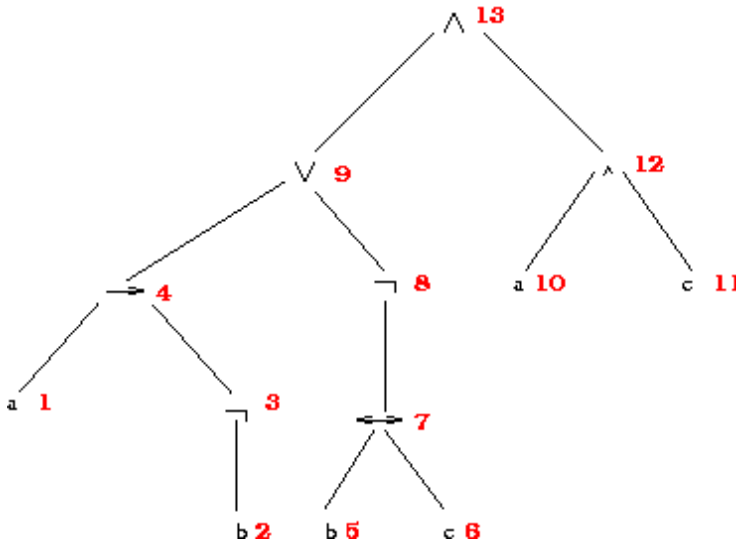
- ' , ' ' , ').

C ' E_1, ' :



- ' , ' ' , ').

C ' E_1 ' :



Question : donner les formes polonaises préfixes et suffixes de l'expression E_0 :

Question : donner les formes polonaises préfixes et suffixes de l'expression E_4 : ⊕

Question : donner les formes polonaises préfixes et suffixes de l'expression E_5 :
⊕

Tables de vérité

- ' ' , , ' - - 2^n) ' .
.

' . '

' F , V.

0, 2^n-1 F 0 V 1),

' ' .

, ' ' , ' - -

. ' - '

.

A ' E_1, :

			$\neg b$	$a \supset \neg b$	$b \equiv c$	$\neg\beta$	$\alpha \vee \gamma$	$a \barwedge c$	E_1 : $\delta \wedge \epsilon$

- **tautologie**

A , ⊃ ⊃ ⊃ _____). □

- **antilogie**)

Question : les formules suivantes sont-elles des tautologies, des antilogies ou ni l'un ni l'autre

∧ ⊃
⊃ ⊃
∧ ⊃ ∧

'_____.

Quelques propriétés et simplifications

-) :
- :

∨
∧

- ∧
∧
∧

- ∨
∨
∨

- **commutativité** :

'

∀ a, b

' ' V ⊃ F F ⊃ V '

).

- **associavité** :

∀ , , ,

' ' :(F ⊃ V)⊃ F F F ⊃(V ⊃ F) T.

- **absorption** :

∧ ∨
∨ ∧

- **distributivité** :
  - a ∨ (b ∧ c) (a ∨ b) ∧ (a ∨ c)
    / :a + (b.c) (a + b).(a + c))
  - a ∧ (b ∨ c) (a ∧ b) ∨ (a ∧ c)

/ :a.(b + c) (a.b) + (a.c))

Question : vérifier que l'opération NAND n'est pas distributive par rapport à l'opération NOR, c'est-à-dire que les expressions

et

ne sont pas équivalentes

- **lois de Morgan** : , .
  :
  - '))
    ∧
  - '))
    ∧ ∨

Question : montrer que les lois de Morgan se généralisent aux opérateurs NAND et NOR, c'est-à-dire que

est équivalente à
est équivalente à

Quels connecteurs choisir

- ' 4 , 16 , 256 , ...
  .
  ' ' -
  '
  .
  complet '
  .
- ' {¬, ∧, ∨} .
  ' '
  .
  forme normale (canonique) disjonctive ,
  $\bigvee_{i=1}^{m}(\bigwedge_{j=1}^{n}\pi_j)$ ' - -).

: **minterme**.

, 2^n .

,

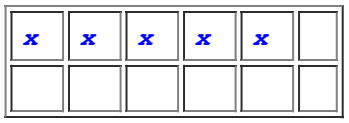
:

- 1,..., n
-)
-)
- ,).

' .

- ,
- , . C ' V F

A ' F 5



:

$$x . x . \overline{x} . x . \overline{x}$$

,

.

- ,

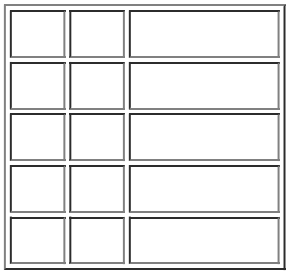
) F' F .

C ' ')

.

Exemples

- ' NOR :



C ' V.

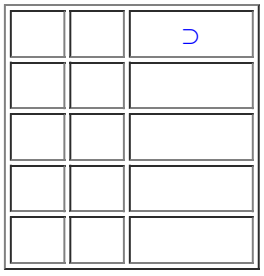
)

' ' '

).

- '

⊃



:

Remarque : ' . ' ⊃

Se pose donc le problème de la simplification

Question : donner la forme normale disjonctive de l'expression
$E_1 : ((a \supset (\neg b)) \vee (\neg(b \equiv c))) \wedge (a \barwedge c)$

Question : donner la forme normale disjonctive de la formule

Question : exprimer l'opérateur

- ' ' e_1 && e_2, ' e_1 false ' e_2 ' : , false.
  ' ' e_1 & e_2, e_1 e_2 .

  Intérêt :
  a b ' ' c ') d. '

  '

) .

- ' ' e_1 || e_2, ' e_1 true ' e_2 ' : , true.
  ' ' e_1 | e_2, e_1 e_2 .

  Intérêt : , ' t ' i ' ' ' t[i] c. '

  '

) i ' .

- **Les opérateurs de manipulation de bits**

  types entiers.

  ' ' .

  - ***les opérations bits à bits***

    ' ,)
    .

    A '

```
int x = 100;
int y = ~x;
System.out.println(x); // system.out.println(x) est équivalent à
Deug.println(x)
System.out.println(y);
```

valeur	

obtenue	explication
-101	32 100 00000000000000000000000001100100 ' :11111111111111111111111110011011 32 -101

),)

)

:

int n = 0x12345678 & 0x90012351 : 32 0001 0000 0000 0000 0000
0010 0101 0000 268436048)

int n = 0x12345678 | 0x90012351 : 32 1001 0010 0011 0100 0111
0111 0111 1001 -1841989767)

int n = 0x12345678 ^ 0x90012351 : 32 1001 0010 0011 0101 0111
0101 0010 1001 -2110425815)

- ***les décalages de bits***

C :
' '

nom de l'opération	notation	commentaire
		. . ' 0 ' , ' .
		. .
' 0 à gauche		. 0 .

:

- 113 32 00000000000000000000000001110001
- -113 32 11111111111111111111111110001111

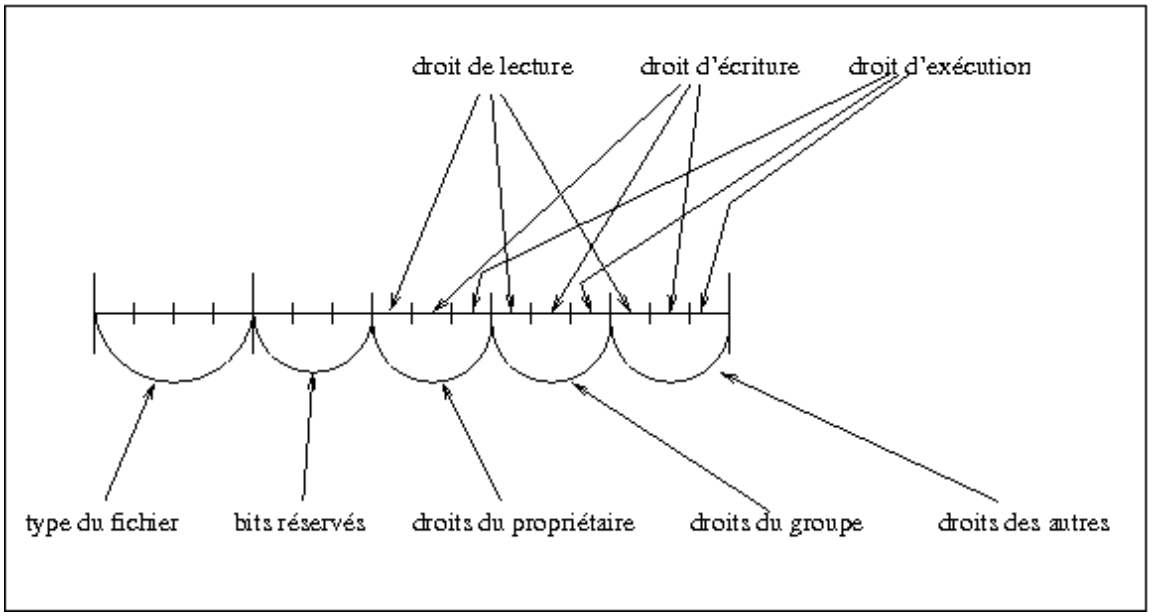
```
int m = 113;
System.out.println(m << 1);
```

```
System.out.println(m >> 1);
System.out.println(m >>> 1);
m = -113;
System.out.println(m << 1);
System.out.println(m >> 1);
System.out.println(m >>> 1);
```

:

valeur obtenue	explication
226	113 << 1 32 : 00000000000000000000000011100010 0 0 32 226 113 × 2)
56	113 >> 1 32 : 00000000000000000000000000111000 0) . 1 32 546 113 / 2)
56	113 >>> 1 32 : 00000000000000000000000000111000
-226	-113 << 1 32 : 11111111111111111111111100011110 ' 226 -113 × 2)
-57	113 >> 1 32 : 11111111111111111111111111000111 , 1 ' -57
2147483591 -113 << 1 32 :01111111111111111111111111000111 0 , 1 ' 2147483591	

Exercice : les systèmes UNIX (par exemple Linux) utilisent un entier de type pour coder à la fois le type du fichier sur 4 bits et les droits des utilisateurs sur le fichier par les 12 autres bits. De manière plus précise, ces 16 bits sont découpés comme suit :



- short
  1010, 000
  111, 101 001.
- '
  4 3)
- true 4
  0110 '

C C