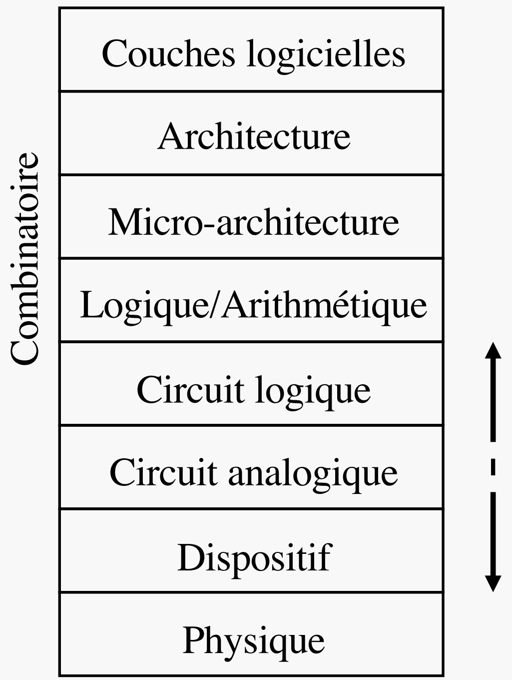
LOGIQUE COMBINATOIRE :
SUITE ET ~FIN

ES102 / CM4

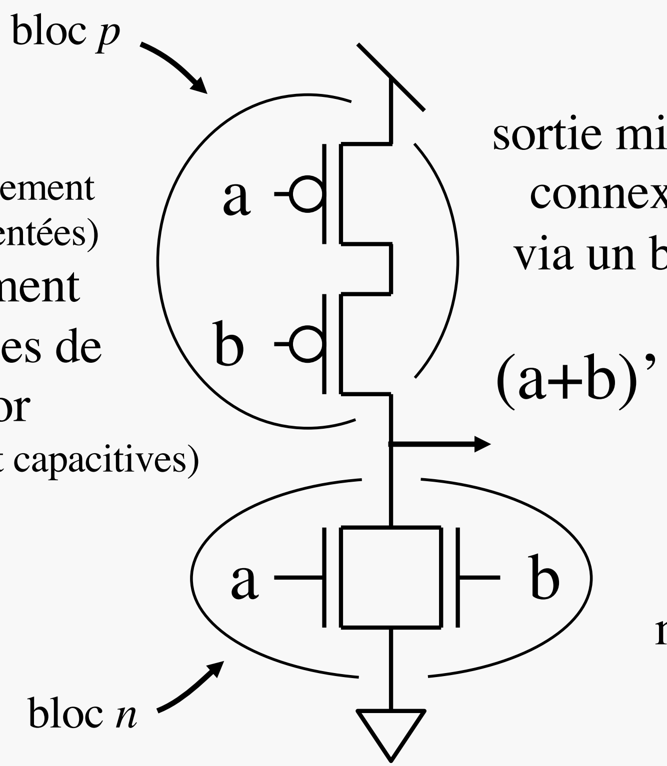


LOGIQUE CMOS

ES102 / CM4a

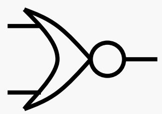
Première expérience CMOS en CM3 & PC3
Recherche d'automatisation et d'optimalité désormais

entrées
(éventuellement complémentées)
uniquement sur grilles de transistor
(purement capacitives)

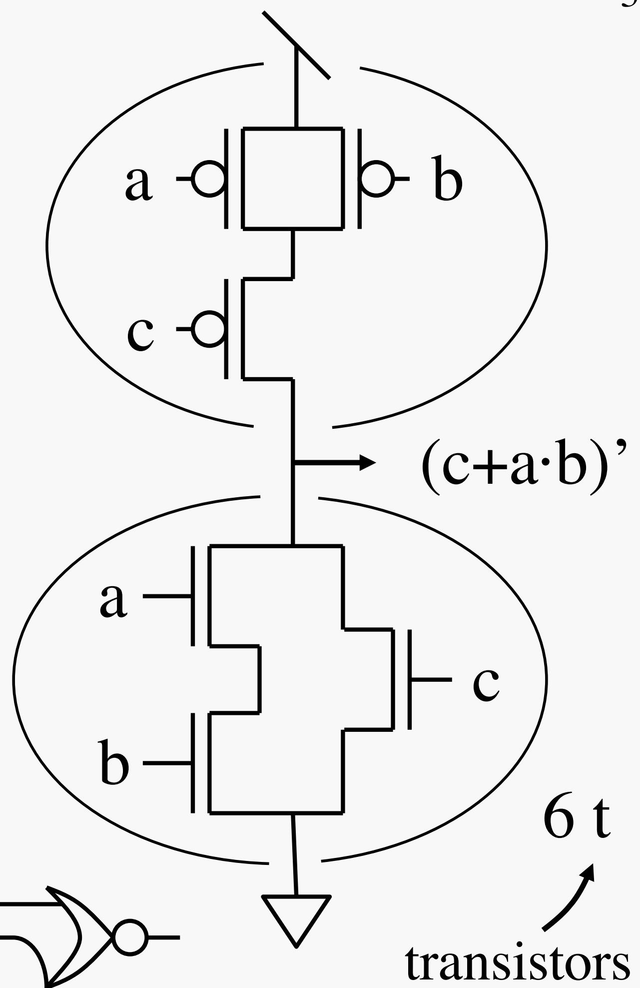


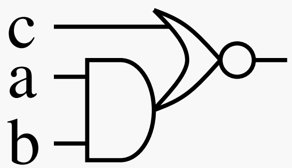
sortie mise à 1 *uniquement* par connexion à l'alimentation via un bloc (de transistors) *p*

sortie mise à 0 *uniquement* par connexion à la masse via un bloc (de transistors) *n*



NOR2

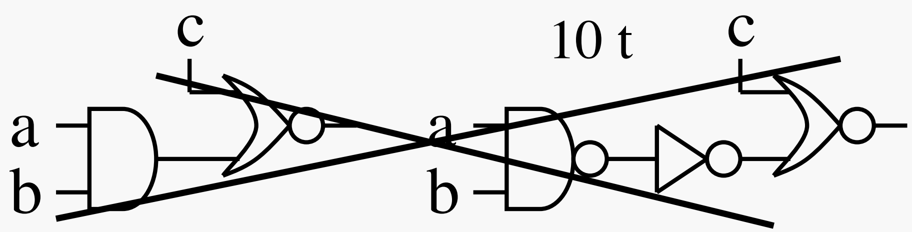




NOR2AND2 : porte *complexe*

LOGIQUE CMOS :

EXEMPLES
ET CARACTÉRISTIQUES



MOTIVATIONS

- Intérêt du *niveau transistors* par opposition au *niveau portes* :
  - accès à la réalité physique, qui détermine les performances
  - accès à des solutions plus compactes et/ou plus rapides
  - le transistor comme unité de mesure de la complexité matérielle

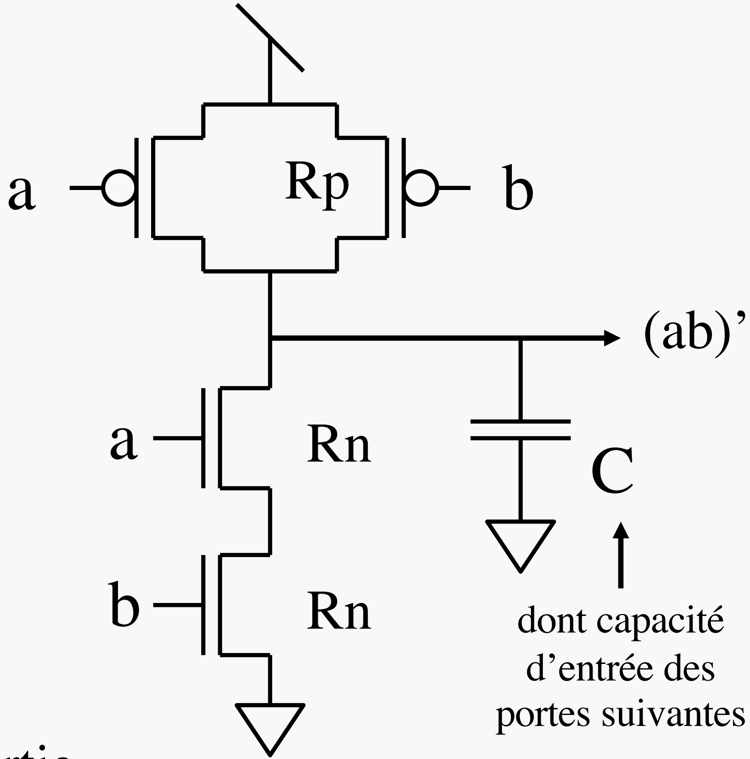
- Compacité :
  - moins de transistors et de connexions au total → NOR2AND2 planche précédente
- Rapidité
  - constantes RC ou Q/I, avec des non-linéarités
  - capacités et résistances moindres puisque gain en compacité
  - mais attention aux résistances :
    - qui se cumulent dans les séries de transistors séparant sortie et masse/alimentation …
    - qui sont nativement plus fortes côté p car inversement proportionnelles à la mobilité μ_p (rappel : $\mu_p \approx 1/3\ \mu_n$) → NAND2 vs NOR2 ↵

VITESSE NAND2 & NOR2 : n VERSUS p

- NAND2 : porte NAND à 2 entrées
- La porte NAND2 commute en sortie
  si une entrée commute, l'autre valant 1
  - ⇒ 1 pMOS contre 2 nMOS en série
- La résistance :
  - entre sortie et alimentation est Rp
  - entre sortie et masse est 2Rn
- Mais $R \propto 1/\mu \quad \Rightarrow \quad Rp \approx 3Rn$

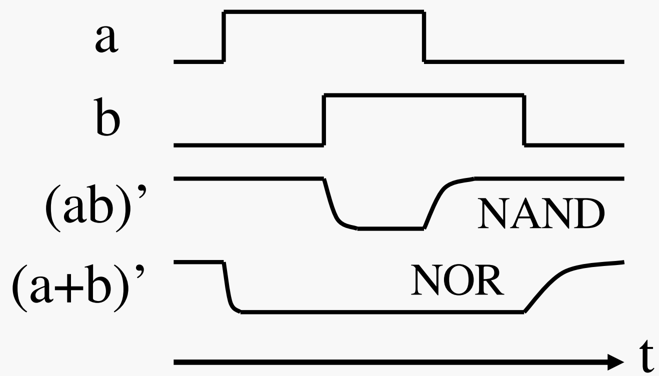
~

⇒ NAND2 naturellement assez équilibrée :
impose un 1 presque aussi vite qu'un 0 sur sa sortie
avec des transistors *n* et *p* de mêmes dimensions



- NOR2 en revanche déséquilibrée d'un facteur ≈ 6
  → pour l'équilibrer, il faut fortement élargir
  les 2 pMOS (canaux *p* plus larges)
  - ⇒ encombrement et capacité d'entrée accrus :-(

Plus de précisions ultérieurement



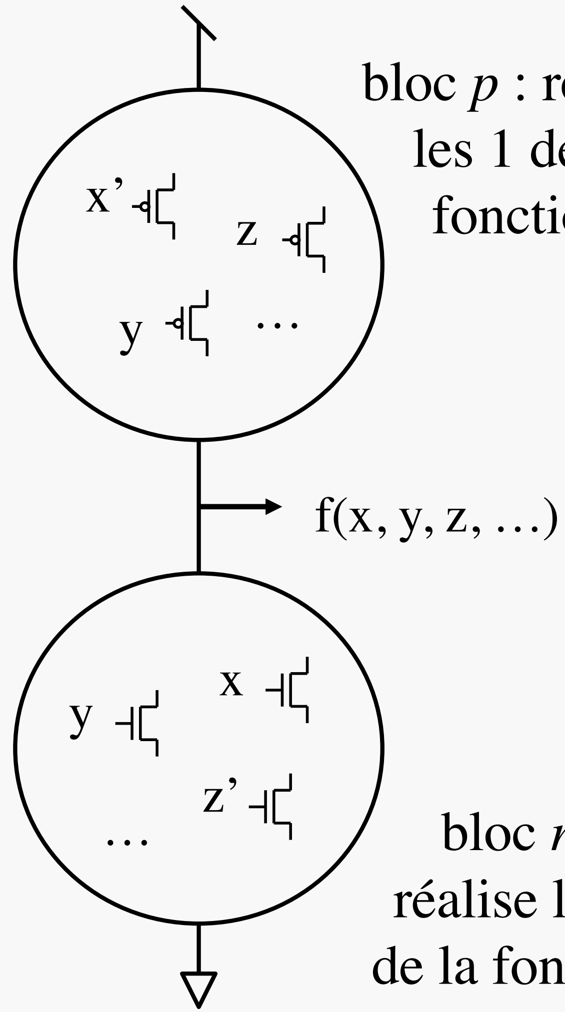
chronogramme

LOGIQUE CMOS : PROBLÉMATIQUE

Analyse :
quelle est la fonction
réalisée par un
montage donné ?

f(x, y, z, ···)

Synthèse :
quel montage pour implanter
une fonction *f* donnée ?
Tout en optimisant
vitesse et compacité …



bloc *p* : réalise
les 1 de la
fonction

bloc *n* :
réalise les 0
de la fonction

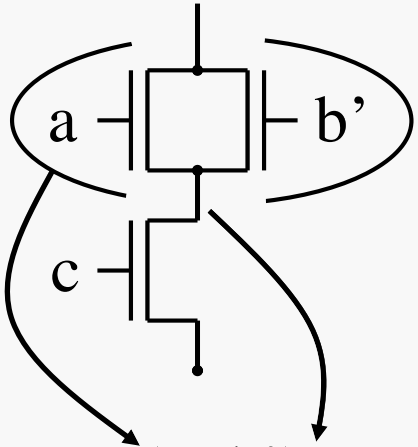
montage = structure...

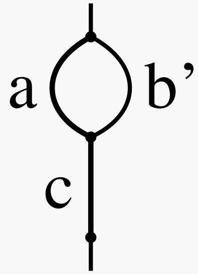
FORMULES STRUCTURELLES γ …

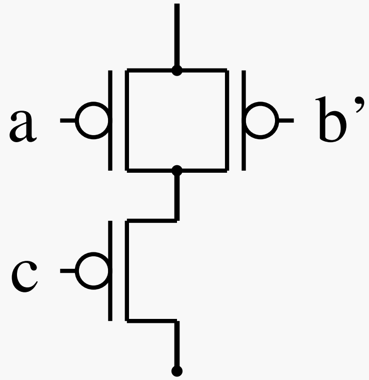
littéral = variable, complémentée ou pas

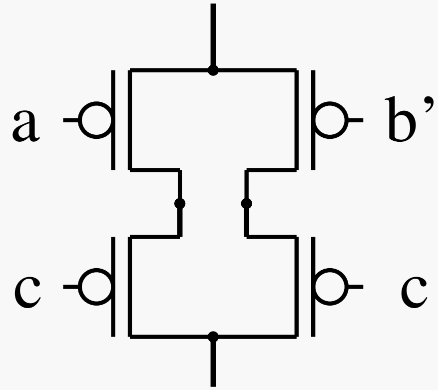
- Blocs (n ou p) → graphes à arêtes valuées par des *littéraux*
- But : les décrire par des formules (ignorant le type de transistor) :
  $\cdot \Leftrightarrow$ mise en série $+ \Leftrightarrow$ mise en // ( ) $\Leftrightarrow$ groupement
- Ex. de formules structurelles, γ_n pour bloc n, γ_p pour bloc p :
  1 transistor ↔ 1 arête ↔ 1 littéral

complément sur variable uniquement !

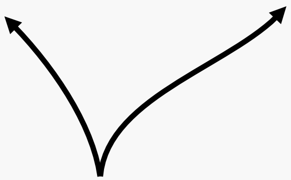


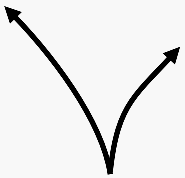






γ_n=(a+b')·c γ_p=(a+b')·c γ_p= ac+b'c





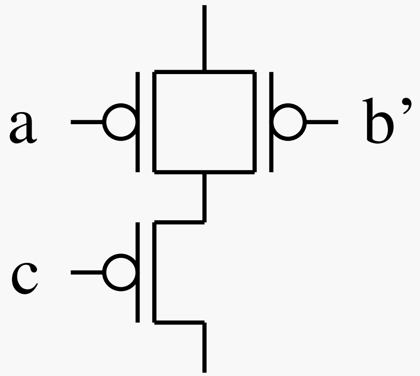
Formules différentes ⇔ structures différentes

Même structure et mêmes commandes ⇔ même formule (le type de transistor n'importe pas)

Mais, interprétées comme formules booléennes, elles correspondent ici à une même fonction ⇔ structures *fonctionnellement* équivalentes

… ET ÉTATS PASSANTS

- L’état passant d’un bloc *n*, resp. *p*, est la condition logique sous laquelle ce bloc connecte la sortie à la masse, resp. l’alimentation
  - il s’agit d’une fonction booléenne
- Formules structurelles γ et états passants (EP) sont liés.
- Interprétant les γ comme formules – puis fonctions – booléennes :
  - $EP_n = \gamma_n$ et $EP_p = \gamma_p/$
    - où ‘/’ = barre de complémentation « vectorisée » : elle complémente chaque variable de γ_p ici



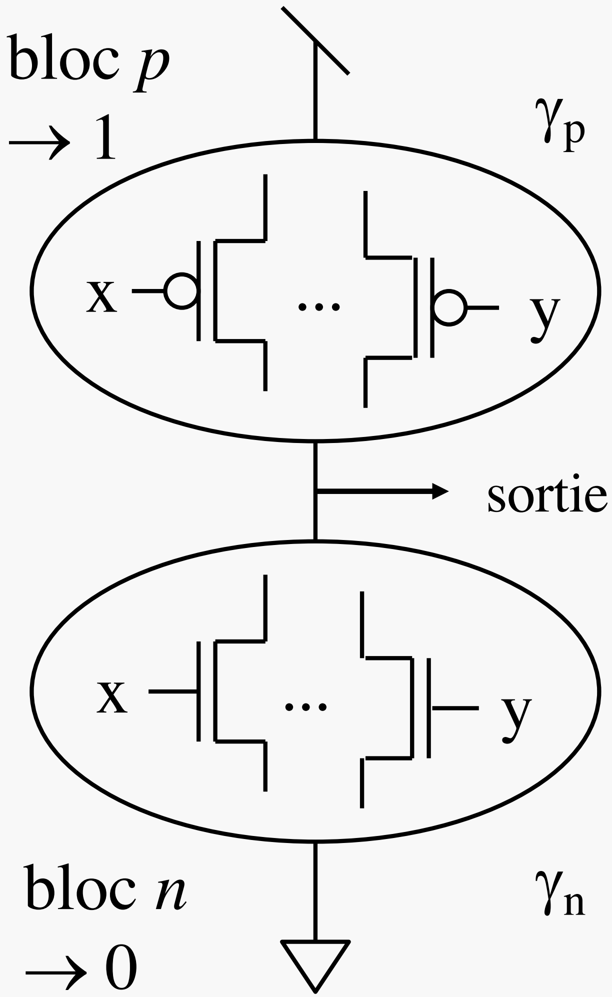
$$\gamma_p = (a+b')\cdot c$$

$$EP_p = (a'+b)\cdot c'$$

$$EP_p = \gamma_p/$$

Lois de De Morgan exprimées avec cette notation :
$/\cdot = +/$ et $/+ = \cdot/$

ANALYSE : $\gamma \rightarrow f$



variables complémentées

état passant : $\gamma_p /$ $\Rightarrow$ $f = \gamma_p /$

un bloc *p* de formule struct. γ_p est passant ssi $\gamma_p/ = 1$ car un pMOS est passant si sa grille est à 0

valeur complémentée

état passant : γ_n $\Rightarrow$ $/f = \gamma_n$ $\Rightarrow$ $f = /\gamma_n$

Un bloc *n* de formule struct. γ_n est passant ssi γ_n=1

Le bloc *n* fournit les 0 de f, c'est-à-dire /f

SYNTHÈSE : $f \rightarrow \gamma$

- Réciproquement à l'analyse : $\gamma_p = f/$ et $\gamma_n = /f$ $(=\overline{f})$
  - mais égalités purement fonctionnelles
  - quelles form(ul)es choisir pour γ_p et γ_n ?

littéral : variable, complémentée ou pas, apparaissant dans une formule

a'b+ab' contient 4 littéraux

- Souhaits :
  - compacité : nombre minimal de transistors
    $\Leftrightarrow$ nombre minimal de littéraux $\rightarrow$ forme **minimale**
  - vitesse : nombre minimal de transistors en série
    $\Rightarrow$ formule avec petits produits $\rightarrow$ forme **disjonctive** minimale (FDM)
  - ainsi, les formules γ_n et γ_p sont idéalement les FDM de leur fonction :

$$\boxed{\gamma_p = \mathrm{FDM}(f/) = \underbrace{\mathrm{FDM}(f)}_{\text{décrire les 1 de f …}}/} \qquad \boxed{\gamma_n = \underbrace{\mathrm{FDM}(/f)}_{\text{décrire les 0 de f …}}}$$

« / sur les variables » commute avec la FDM car simple symétrie centrale sur le n-cube

… sous la forme $\Sigma\Pi$ la plus compacte

SYNTHÈSE : EXEMPLE

① $f = ac' + a'b'c'$

② $\gamma_p = FDM(f)/$

③ $FDM(f) = ac'+b'c'$

④ $\gamma_p = a'c+bc$

⑤

4 littéraux ⇔ 4 transistors

f			a	a
	1	0	1	1
c	0	0	0	0
		b	b	

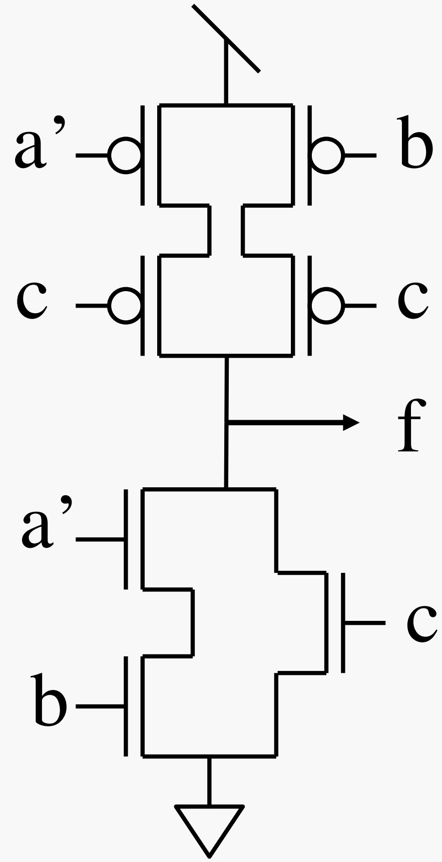
⑥ $\gamma_n = FDM(/f)$

⑦

⑧ $FDM(/f) = a'b+c$

⑨ $\gamma_n = a'b+c$

⑩



3 littéraux ⇔ 3 transistors

⑪ + 1 inverseur nécessaire pour fournir a'

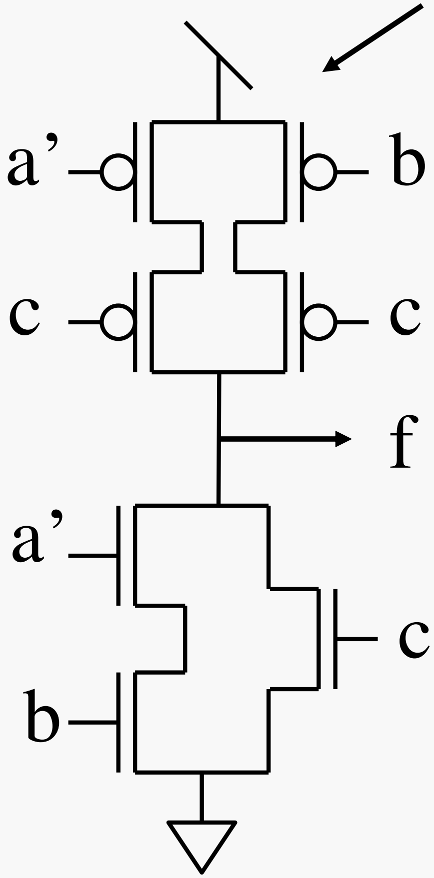
AU-DELÀ DE LA FDM…

Factoriser 2 termes d'une $\Sigma\Pi$ fournit en général une $\Sigma\Pi\Sigma\Pi$, qui dégénère ici en $\Pi\Sigma$ (la FCM en fait).

bi-niveau

multi-niveau

γ_p = FDM(f)/
= a'c+bc

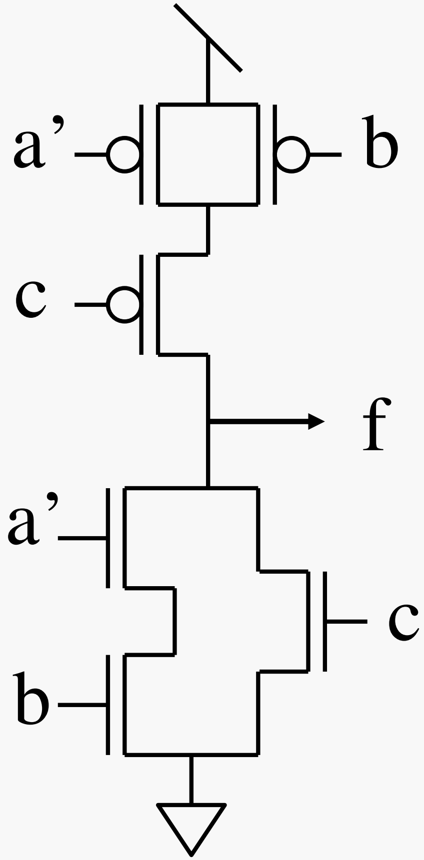


factorisation

γ_p = (a'+b)c

dualité… ?

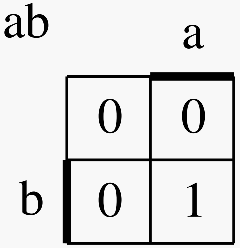
γ_n = FDM(/f) = a'b+c



Dualité : γ_p et γ_n échangent · et + $\Leftrightarrow$ blocs *p* et *n* échangent structures série et parallèle.
Propriété se limitant en fait aux portes les plus simples → PC4.

LOGIQUE CMOS
LIMITATION 1 : DÉCROISSANCE

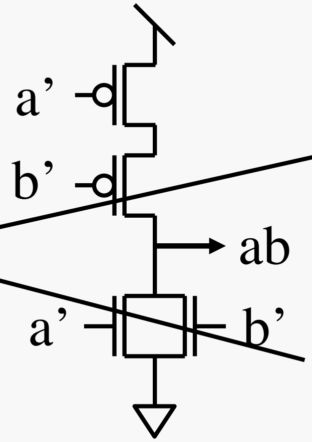
$f(a, b)$
$= ab$

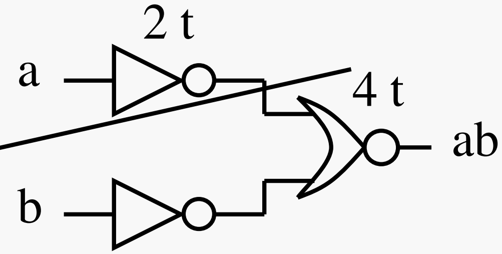


Exemple : synthèse automatique du ET logique

γ_p = FDM(f)/ = a'b'

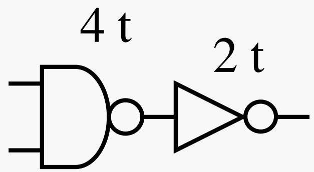
γ_n = FDM(/f) = a'+b'





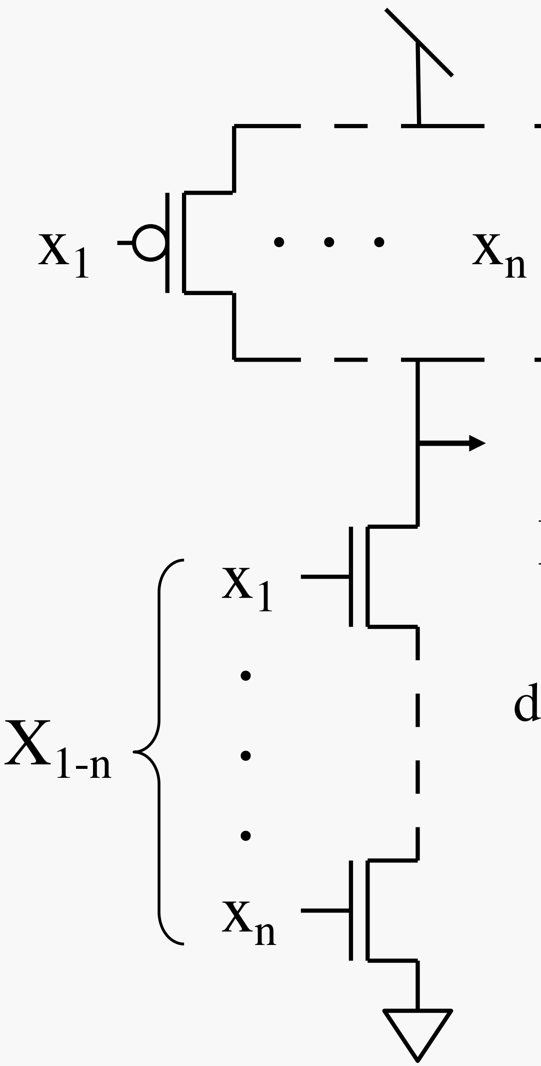
Restitution au niveau portes

Meilleure solution :



- Implanter une fonction en logique CMOS nécessite en général de complémenter de multiples variables, au moyen d'inverseurs placés en amont, ce qui augmente surface et délai
  mais cela reste avantageux par rapport aux solutions en portes élémentaires, sauf cas élémentaire…
- Les fonctions décroissantes échappent à cette contrainte
  d'où des portes particulièrement compactes et rapides ☺
- Un cas particulier est celui des fonctions croissantes, dont il vaut mieux implanter la complémentée, suivi d'un inverseur, pour gagner en transistors ☺

LOGIQUE CMOS
LIMITATION 2 : ENTRANCE

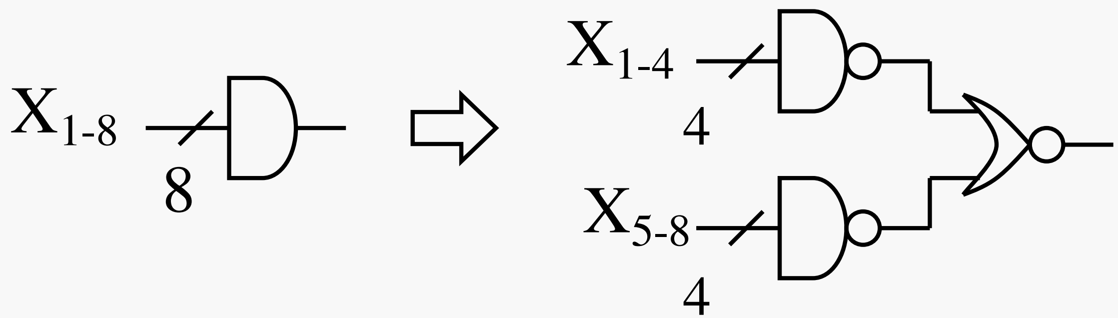


NANDn

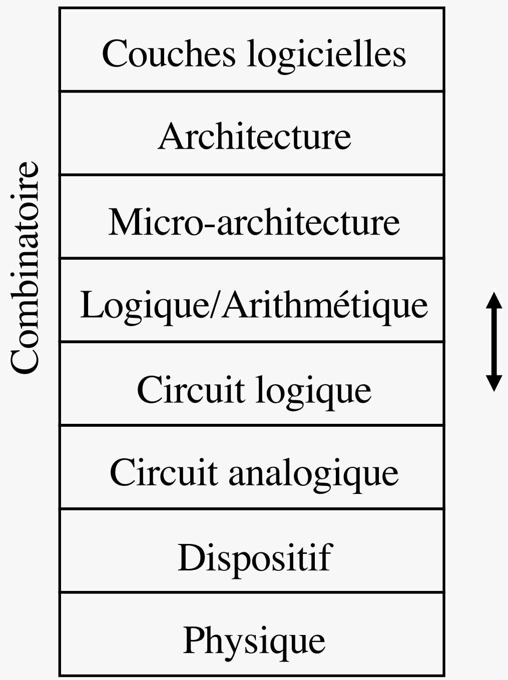
une porte d'entrance *n*

- Entrance = nombre d'entrées d'une porte
- Si trop grande, risque de longues séries de transistors ⇒ délais excessifs, en $O(n^2)$
  - typiquement pour $n \geq 5$

→ Décomposition préalable nécessaire (commode à représenter au niveau portes) : « fragmentation » en portes plus légères (au moyen d'une méthode *ad hoc*)



→ **besoins de collaboration bien comprise entre niveaux portes et transitors**



FDM : FORMALISATION & ALGORITHMES

ES102 / CM4b

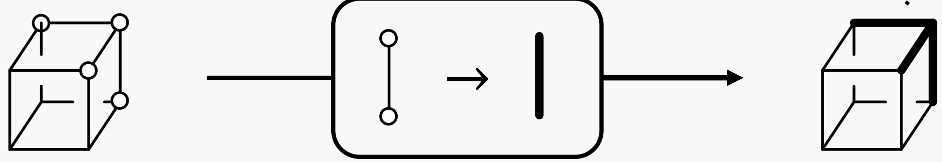
FORME DISJONCTIVE MINIMALE (FDM)

- Importance confirmée. Obtenue par méthode :
  - graphique/géométrique, pour s'initier avec *n* petit
  - algébrique/algorithmique/programmée, pour cas général & CAO

- Procédé :

sous-cubes

1-sous-cube



$\underline{f}$

a'bc+ab'c
+abc'+abc

fusions

x hors π

ab+bc+ac

FND(f) — $x'\pi + x\pi \rightarrow \pi$ → FDM(f)

forme $\Sigma\Pi$ où les produits sont des mintermes

produits de littéraux

forme $\Sigma\Pi$ à nombre minimal de littéraux

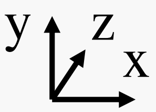
Les produits mis en jeu pour passer de FND à FDM seront appelés « implicants » de f

implicant premier

abc $\Rightarrow$ ab $\Rightarrow$ Maj(a, b, c)

SOUS-CUBES DE $\mathbb{B}^n$

- Généralisation en dimension *n* des sous-ensembles du type sommets, arêtes et faces du 3-cube $\mathbb{B}^3$



		face	arête	sommet
		2-sous-cube	1-sous-cube	0-sous-cube
variables fixes/libres $\longrightarrow$	– – –	– 0 –	0 – 1	1 1 0
fonction indicatrice $\longrightarrow$	1	y’	x’z	xyz’

- L’indicatrice d’un sous-cube de $\mathbb{B}^n$ est un produit de *m* littéraux
  - mettant en jeu *m* variables distinctes (parmi les *n* décrivant $\mathbb{B}^n$) aux valeurs fixées sur le sous-cube considéré
  - les n-m autres variables sont libres :
    - (n-m) constitue la dimension du sous-cube, et *m* sa *co*dimension
    - on dit que c’est un *(n-m)-sous-cube*
      ~ sous-espace affine de dimension (n-m)
  - $\exists\ 3^n$ sous-cubes dans $\mathbb{B}^n$: $\{0, 1, –\}^n$ $\left\{ \text{chaque variable peut être fixée (à 0 ou 1) ou libre} \right.$

complément à la dimension de l’espace

MINIMISATION : ALGÈBRE ↔ GÉOMÉTRIE

- π produit de littéraux ↔ $\underline{\pi}$ sous-cube
- nombre de littéraux dans π = codimension de $\underline{\pi}$
- π implicant de f ↔ $\underline{\pi} \subset \underline{f}$ ← *support*
- expression $\Sigma\Pi$ (alias FD) ↔ $\cup$ de sous-cubes
- $f = \Sigma\Pi$ ↔ $\underline{f} = \cup$ de sous-cubes ⇔ { sous-cubes } « couvre » $\underline{f}$

couvrir ≠ partitionner : $\cap$ 2 à 2 ≠ ø

- « minimiser » f ↔ couvrir $\underline{f}$ avec un ensemble de sous-cubes à somme des codimensions minimale

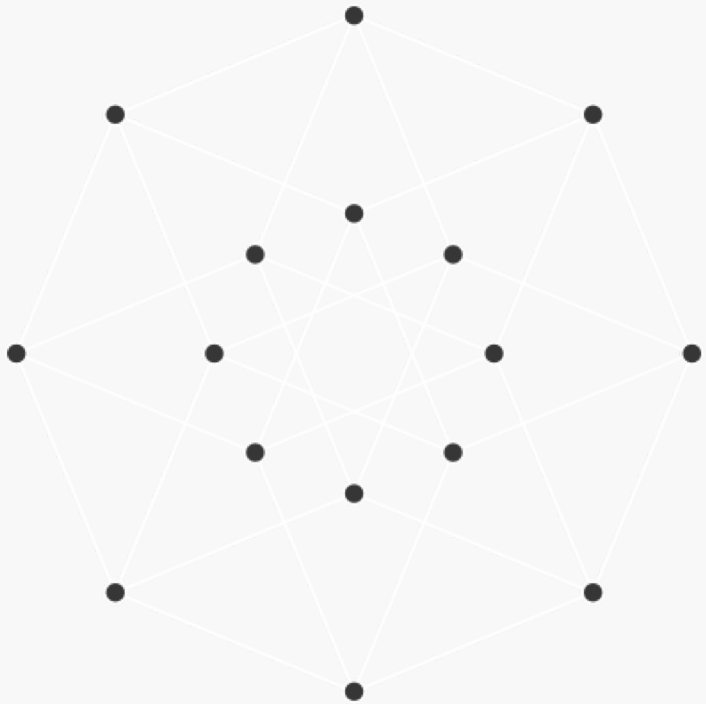
le nb de littéraux dans la FD de

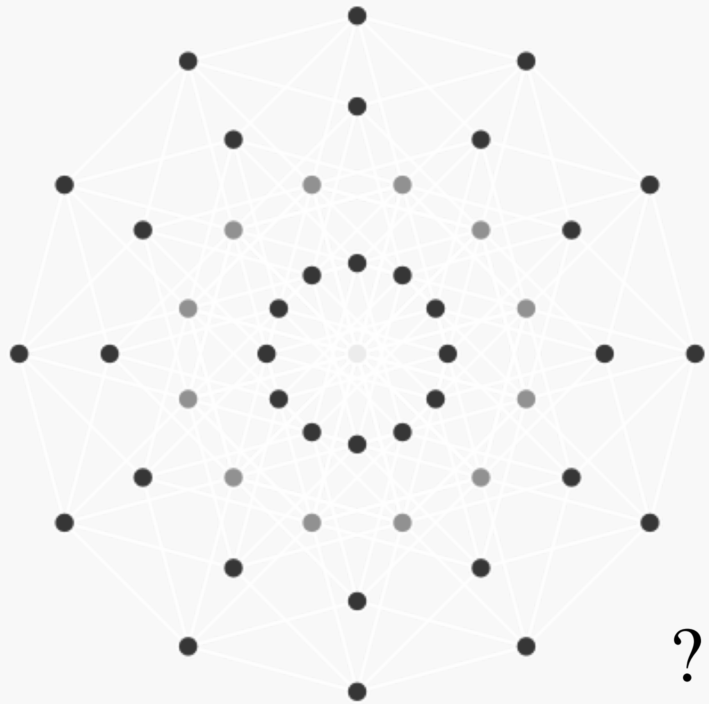
égale au nombre de transistors nécessaires

- π_1 *implicant premier* de f ssi il n'existe pas $\pi_2 \neq \pi_1$ tel que $\pi_1 \Rightarrow \pi_2 \Rightarrow f$ ↔ Γ_1 *sous-cube maximal* dans $\underline{f}$ ssi il n'existe pas d'autre sous-cube Γ_2 tel que $\Gamma_1 \subset \Gamma_2 \subset \underline{f}$
- Th. : *la FDM de f ne comporte que des implicants premiers de f* ↔ Minimiser f ⇒ couvrir $\underline{f}$ avec des sous-cubes maximaux
  - évident par l'absurde
  - mais tous ne sont pas nécessaires !

REPRÉSENTATION 2-D DU n-CUBE ?

- Préserver la topologie du n-cube :
  - des points voisins sur le n-cube doivent le demeurer sur sa représentation
  - exemples avec le 4-cube et le 6-cube (http://fr.wikipedia.org/wiki/Hypercube) :



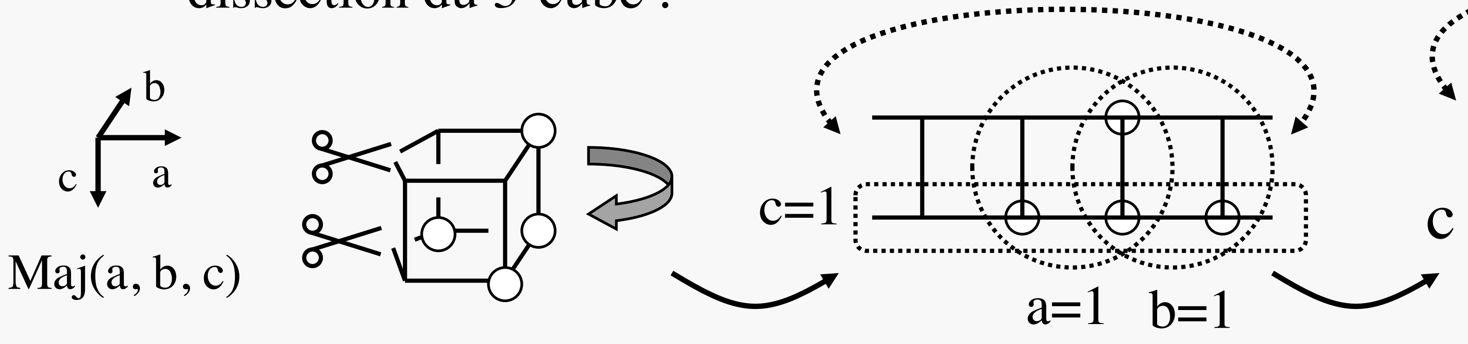


?

- Mais aussi pouvoir y repérer les sous-cubes...

REPRÉSENTATION 2-D DU n-CUBE

- Préserver la topologie du n-cube et y voir les sous-cubes facilement
    - pour pouvoir identifier les sous-cubes maximaux
  - – dissection du 3-cube :



a

	0	0	1	0
c	0	1	1	1

b

trois 1-sous-cubes maximaux bien visibles

  $\rightarrow$ tables de Karnaugh fonctionnelles du CM1 !
    - couple (a, b) suivant le code de Gray 2
    - rebouclage par les bords (périodique 1D)
  - – dissection simple aussi pour n=4 $\rightarrow$ tableau 4×4 torique (périodique 2D) ↵
  - – mais topologie forcément rompue si n≥3 variables sur une dimension :
    - car 12 (nb d'arêtes du 3-cube) > 8 (frontières entre cases de table 1-D)
      - – moindre mal : Gray 3

(a, b, c)

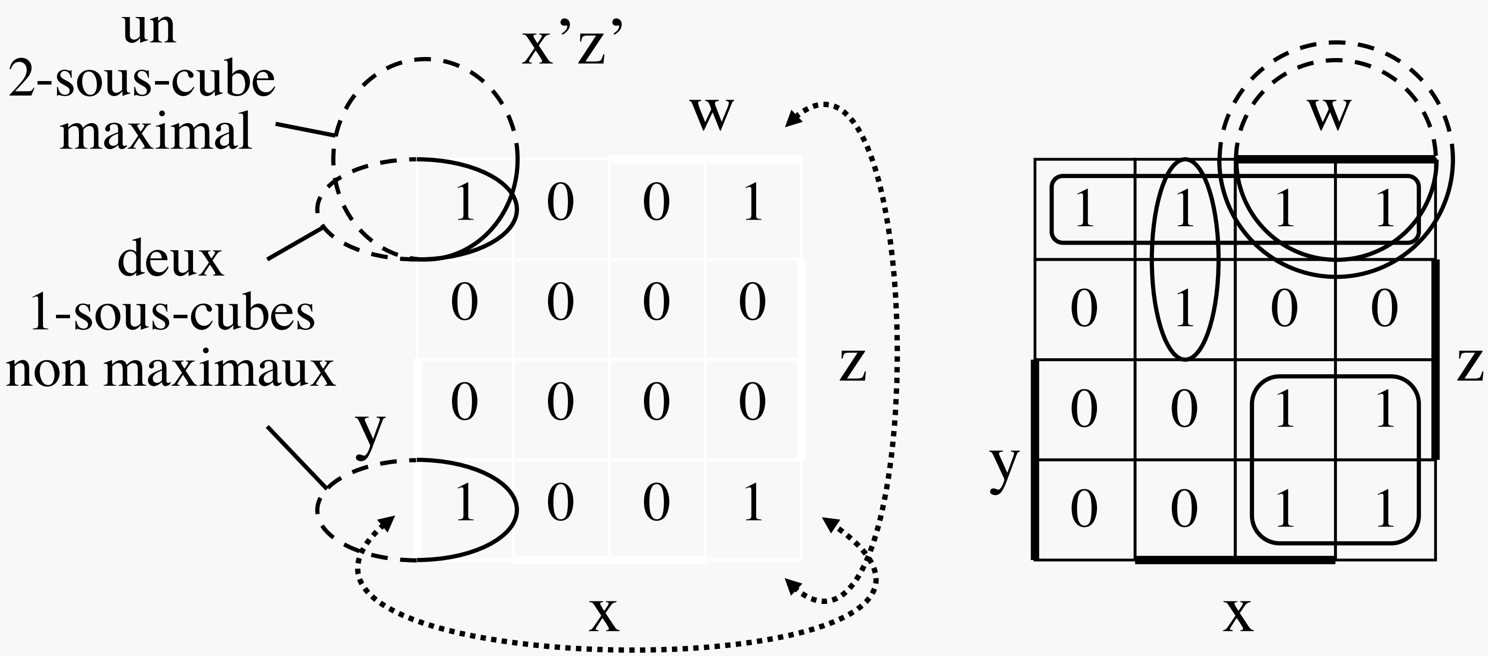
b a

(0, 0, 0)	(0, 0, 1)	(0, 1, 1)	(0, 1, 0)	(1, 1, 0)	(1, 1, 1)	(1, 0, 1)	(1, 0, 0)

c c

MÉTHODE GRAPHIQUE DE KARNAUGH

- Sur table de Karnaugh, sous-cubes = zones rectangulaires dont largeur et hauteur sont des puissances de 2
- La méthode de Karnaugh, graphique, consiste à couvrir *à la main* les *1* de la fonction, par un jeu minimal de sous-cubes maximaux
- Un sous-cube maximal est dit « *essentiel* » s'il est seul à couvrir certain(s) *1* de la fonction. Son implicant premier associé est également dit *essentiel* et il est *forcément dans la FDM*.



$y'z' + w'xy' + \cancel{w}z' + wy$

implicants premiers tous essentiels ici sauf wz', qui s'avère inutile

$= y'(z'+w'x) + wy$

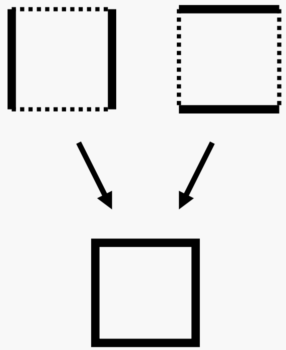
Factorisation post-FDM possible si 2 sous-cubes maximaux dans un même sous-cube (ici, y=0)

MÉTHODE ALGÉBRIQUE DE QUINE-McCLUSKEY

- Comment déterminer les implicants premiers de f *sur ordinateur* ?
  - par construction progressive, partant de la FND (mintermes)
    - ↔ sous-cubes de dimension croissante (points, arêtes, ...)
  - mécanisme : union de paire d'implicants jumeaux à une variable près, complémentée dans l'un mais pas dans l'autre

    $\pi x' + \pi x = \pi$, où π produit de k variables hors x

    - ↔ $\cup$ de deux (n-k-1)-sous-cubes symétriques par rapport à x en un (n-k)-sous-cube



- Algorithme de Quine :
  - former une population initiale constituée de tous les termes de la FND (ce sont des mintermes)
  - réaliser toute union possible
    - chaque minterme peut participer à plusieurs unions
  - recommencer avec leurs descendants, et ainsi de suite
    - jusqu'à épuisement des possibilités
  - les individus, initiaux ou descendants, n'ayant pu participer à aucune union sont les *implicants premiers* (IPs) de f

QUINE : EXEMPLE

dimension des sous-cubes entrant successivement dans la population

nombre de variables non complémentées	0 (mintermes)	1	2	3
0	*w’x’y’z’*	x’y’z’ w’y’z’	**y’z’**	—
1	wx’y’z’ w’xy’z’	wx’z’ xy’z’ wy’z’ **w’xy’**	**wz’**	—
2	wx’yz’ wxy’z’ *w’xy’z*	wz’y wyz’ wxz’	**wy**	
3	wx’yz wxyz’	wyz wxy		
4	*wxyz*			

w

	1	1	1	1
	0	1	0	0
y	0	0	1	1
y	0	0	1	1

z

x

IPs = { y’z’ , wz’, w’xy’, wy }

non essentiel

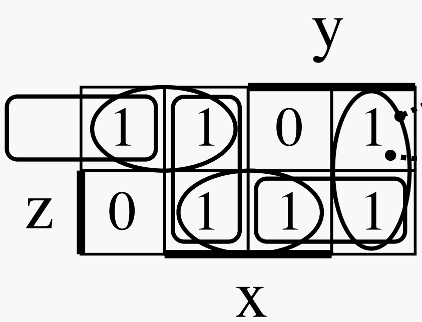
Un minterme (0-sous-cube) qui implique *un seul* implicant premier (sous-cube maximal) le rend *essentiel*

Méthode de Quine facile à programmer, mais jusqu’à 3^n implicants à manipuler : complexité exponentielle

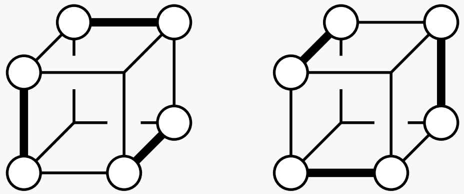
QUELS IMPLICANTS PREMIERS RETENIR POUR LA FDM ?

Implicants Premiers Essentiels

$$\underbrace{\sum \text{IPE}(f)}_{\text{parfois nul !}} \Rightarrow f \Rightarrow \sum \text{IP}(f)$$



situation extrême :
aucun sous-cube
maximal essentiel !



Et ~~la~~ FDM n'est pas unique !

$|\cdot| > 3^n/n$

Implicants premiers non essentiels

1 de la fonction non déjà couverts par un IPE $|\cdot| \sim 2^n$

xyz	x'y 01-	x'z' 0-0	xy' 10-	xz 1-1	y'z' -00	yz -11
000		×			×	
010	×	×				
011	×					×
100			×		×	
101			×	×		
111				×		×

Quel jeu minimal de colonnes sélectionner pour couvrir toutes les lignes ?

Problème algorithmique difficile classique, dit de « *couverture exacte* »

Explosion combinatoire ~Sudoku hors ES102

SYNTHÈSE LOGIQUE : HISTORIQUE

- années 50 : Karnaugh, Quine, McCluskey
  - fournit une/la FDM, mais ne passera pas à l'échelle
  - connaissances fondamentales néanmoins
- années 70 : théorie de la complexité NP

  catégorisation de problèmes algorithmiques difficiles
- années 80 : renoncement à la minimalité
  - *Espresso* de IBM & Berkeley University
  - FD quasi-M par approche heuristique en temps très réduit
- années 90 : accélération par utilisation des BDD

  → *O. Coudert, Doing two-level minimization 100 times faster, 01/1995*
- parallèlement (>80), minimisation multiniveau (~ hors ES102)
  - factorisation et décomposition de fonctions booléennes
- depuis, préoccupations système :
  - face à la complexité des réalisations actuelles (*design gap*)

  → synthèse architecturale, synthèse comportementale, system level design (*SLD*)

grands noms de la CAO industrielle :

cadence

SYNOPSYS Predictable Success

Mentor Graphics

MAGMA

AVANT-GOÛT DE PC4

- commande d'un afficheur de chiffres décimaux à 7 segments
  - chiffre k = $(hqdu)_2$

    huit **q**uatre **d**eux **u**n

- fonction booléenne incomplètement spécifiée :
  - on profite des cas indifférents pour compacter la FDM
    - en plaçant des 1 là où cela arrange
  - exemple : sgb = q'u'+du' (*sgb* : segment gauche bas)