

Introduction à l'informatique

pour les mathématiques, la physique et les sciences
computationnelles

Yann Thorimbert



**UNIVERSITÉ
DE GENÈVE**

CENTRE UNIVERSITAIRE
D'INFORMATIQUE

Chapitre 4

Algèbre de Boole et circuits logiques

Yann Thorimbert



UNIVERSITÉ
DE GENÈVE

CENTRE UNIVERSITAIRE
D'INFORMATIQUE



Chapitres du cours

1. Origines des ordinateurs et des réseaux informatiques
2. Codage des nombres
3. Codage des médias
4. **Circuits logiques** ←
5. Architecture des ordinateurs
6. Conception et exécution de programmes
7. Algorithmique, programmation et structures de données

Rappel

- On peut coder les informations qui nous intéressent sous forme d'états binaires (cf. chapitres précédents).
- On veut réaliser des raisonnements logiques qui portent sur ces données. On doit faire subir un traitement à ces états binaires.

Exemple de table de vérité

- Considérons pour l'exemple un cas simplifié d'infraction routière en Suisse :

"Une personne est en excès de vitesse si elle roule à plus de 50 km/h à l'intérieur d'une localité."

- Nous voulons qu'un ordinateur puisse traiter ce type de **proposition logique**.
- Pour formaliser une telle phrase, on peut :
 - 1) Repérer les sous-parties : *être en excès de vitesse*, *être hors localité*, ...
 - 2) Résumer le tout sous la forme d'une **table de vérité**.



Exemple de table de vérité

Limitation de vitesse dans les localités en Suisse (où 1 = "oui", 0 = "non").



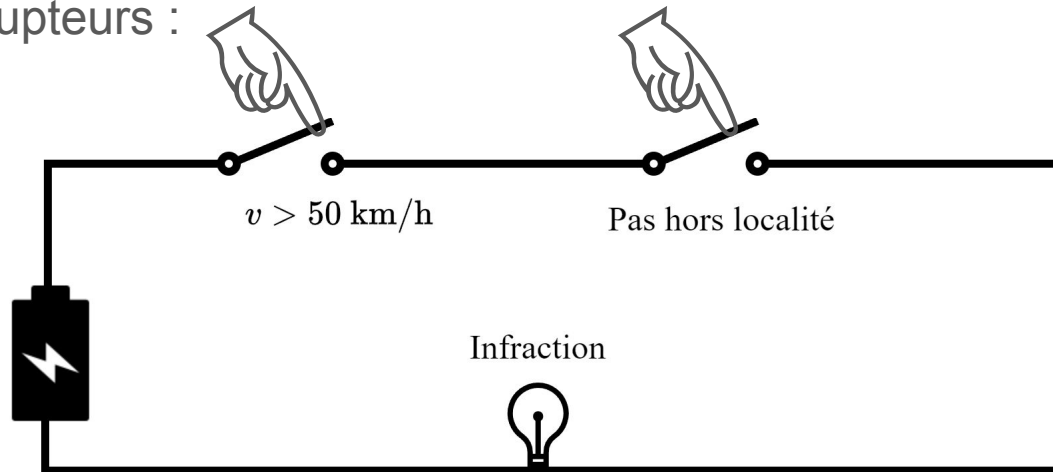
Les éléments d'une expression logique

"Est en infraction" = "Roule à plus de 50 km/h" ET PAS("Hors localité")

De table de vérité à circuit logique

"Est en infraction" = "Roule à plus de 50 km/h" ET PAS("hors localité")

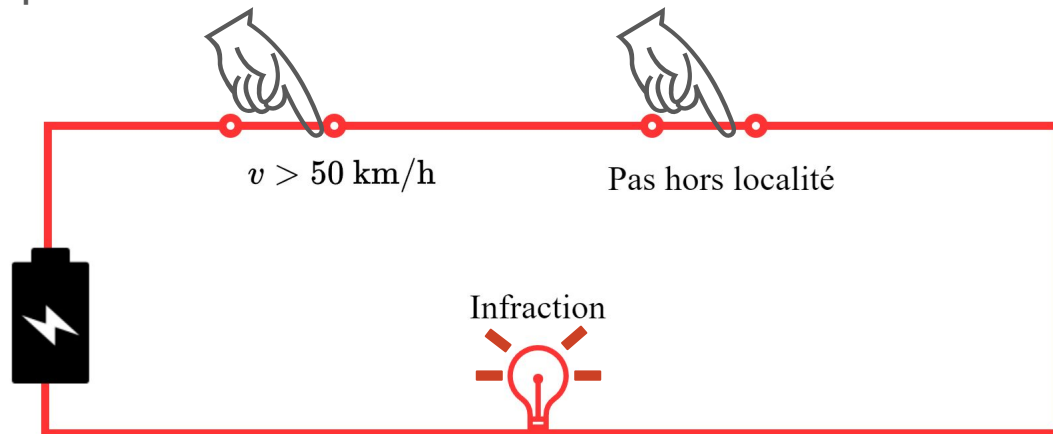
On pourrait **réaliser** cette fonction logique sous la forme d'un circuit électrique avec des interrupteurs :



De table de vérité à circuit logique

"Est en infraction" = "Roule à plus de 50 km/h" ET PAS("hors localité")

On pourrait **réaliser** cette fonction logique sous la forme d'un circuit électrique avec des interrupteurs :



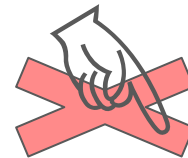
De table de vérité à circuit logique

"Est en infraction" = "Roule à plus de 50 km/h" ET PAS("hors localité")

On pourrait réaliser cette fonction logique sous la forme d'un circuit électrique avec des interrupteurs.

Cependant, on veut **automatiser** le processus.

De plus, on veut **abstraire** le processus.



De table de vérité à circuit logique

"Est en infraction" = "Roule à plus de 50 km/h" ET PAS("hors localité")

- Une table de vérité peut tout aussi bien être résumée par un **schéma**.
- Le schéma doit incorporer les **fonctions logiques** que sont ET et PAS.

De table de vérité à circuit logique

"Est en infraction" = "Roule à plus de 50 km/h" ET PAS("hors localité")

- Une table de vérité peut tout aussi bien être résumée par un **schéma**.
- Utilisons les symboles arbitraires suivants, nommés **portes logiques** :
 - ET est arbitrairement symbolisé par le symbole 
 - PAS est arbitrairement symbolisé par le symbole 

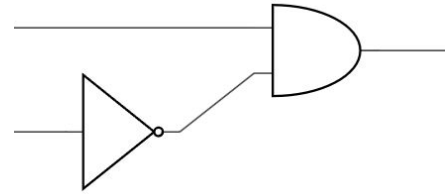
De table de vérité à circuit logique

Roule à plus de 50 km/h	Est hors localité	Est en infraction
0	0	0
0	1	0
1	0	1
1	1	0

De table de vérité à circuit logique

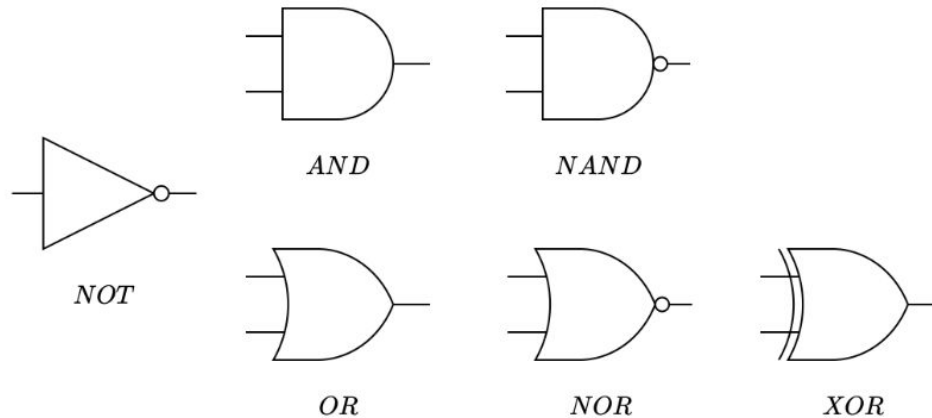
- Table de vérité faite de **propositions logiques binaires**.
- Circuit logique fait de **portes logiques** prenant des valeurs binaires en entrée et retournant des valeurs binaires en sortie.
- Le circuit **réalise** la table de vérité.

Roule à plus de 50 km/h	Est hors localité	Est en infraction
0	0	0
0	1	0
1	0	1
1	1	0



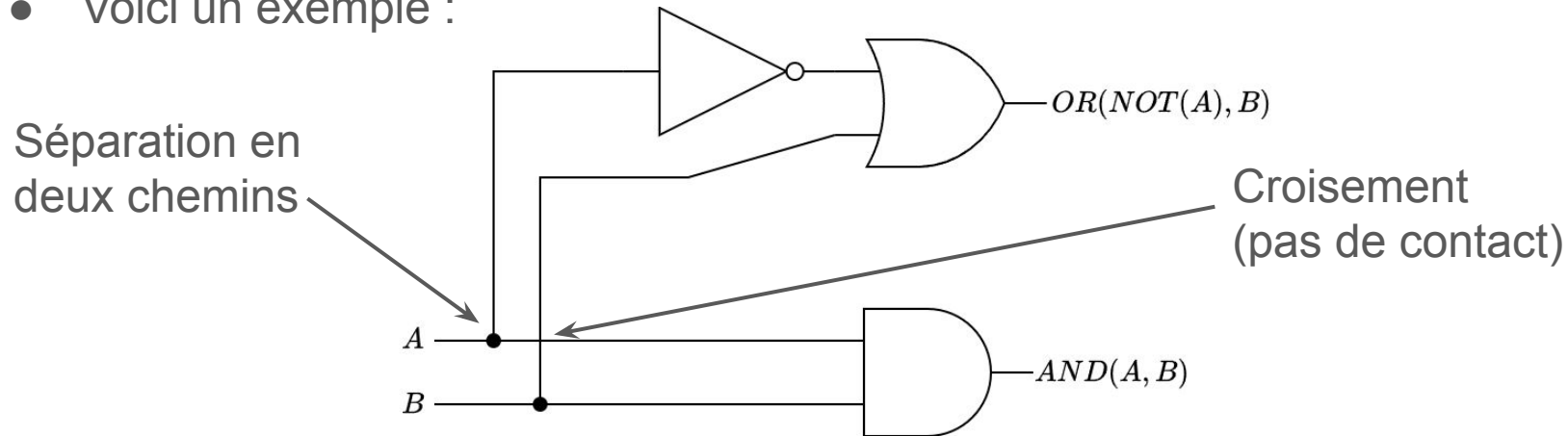
Différents types de portes

Chaque porte correspond à une table de vérité.



Note sur les schémas de circuits logiques

- On représente les noeuds ("séparations" du fil) avec un point.
- On représente les fils qui se croisent sans point.
- Parfois, nous écrirons des expressions sur le schéma.
- Voici un exemple :



Rappel : transistors

- Les transistors sont les éléments électroniques de base des ordinateurs actuels.
- Méthode efficace, mais pas unique (cf. diapos ordinateur à eau) pour émuler des portes logiques avec un système physique.
- Robinets à électrons : laissent (ou pas) passer le courant électrique entre deux bornes en fonction de la tension à une troisième borne.
- Si l'on peut réaliser les schémas dans la vie réelle, alors on peut simuler un raisonnement logique grâce à une machine !



Aparté physique

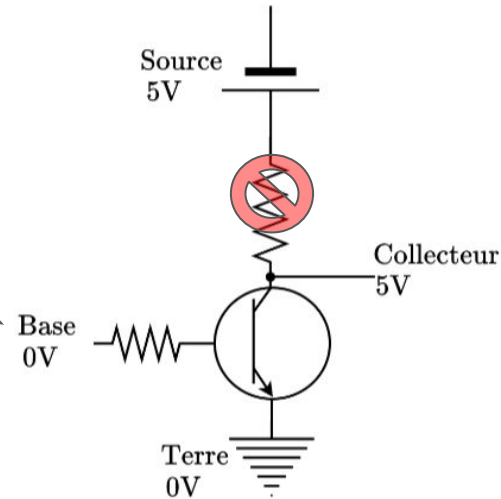
- Quand un courant électrique traverse une résistance, on mesure une différence de tension aux bornes de la résistance.
- Loi d'Ohm : $\Delta U = U_2 - U_1 = R \cdot I$
- On dit que la résistance fait "chuter" la tension.



De transistors à portes logiques

Fonction logique NOT :

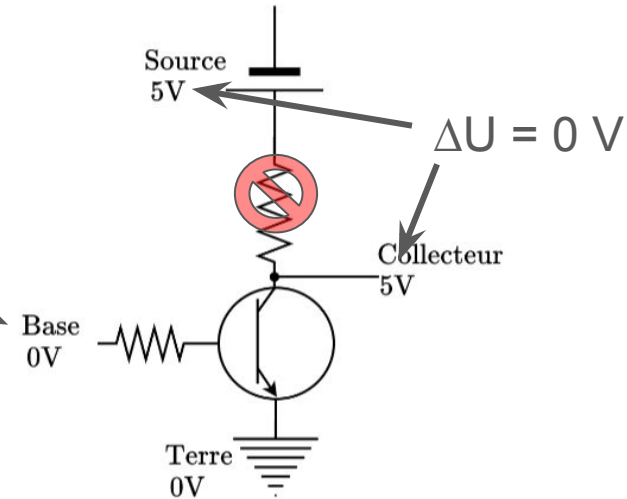
Base **n'active pas** le transistor :
⇒ Le courant ne s'établit pas
⇒ Pas de chute de tension.



De transistors à portes logiques

Fonction logique NOT :

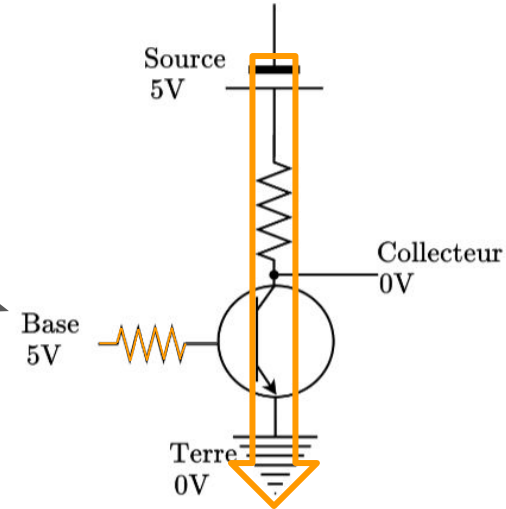
Base **n'active pas** le transistor
⇒ Le courant ne s'établit pas.
⇒ Pas de chute de tension.



De transistors à portes logiques

Fonction logique NOT :

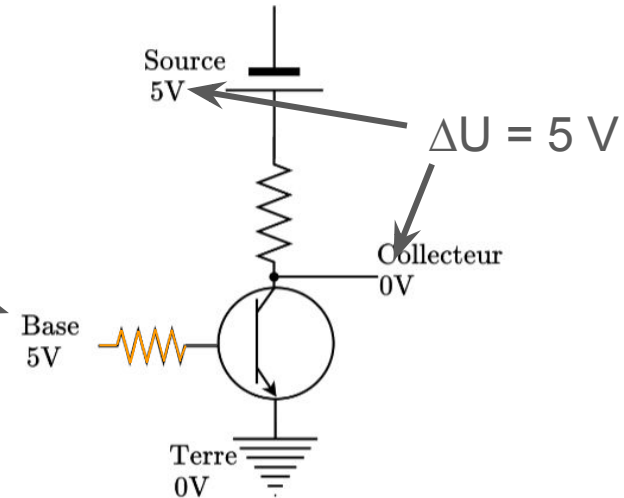
Base **active** le transistor.
⇒ Le courant s'établit.
⇒ Chute de tension.



De transistors à portes logiques

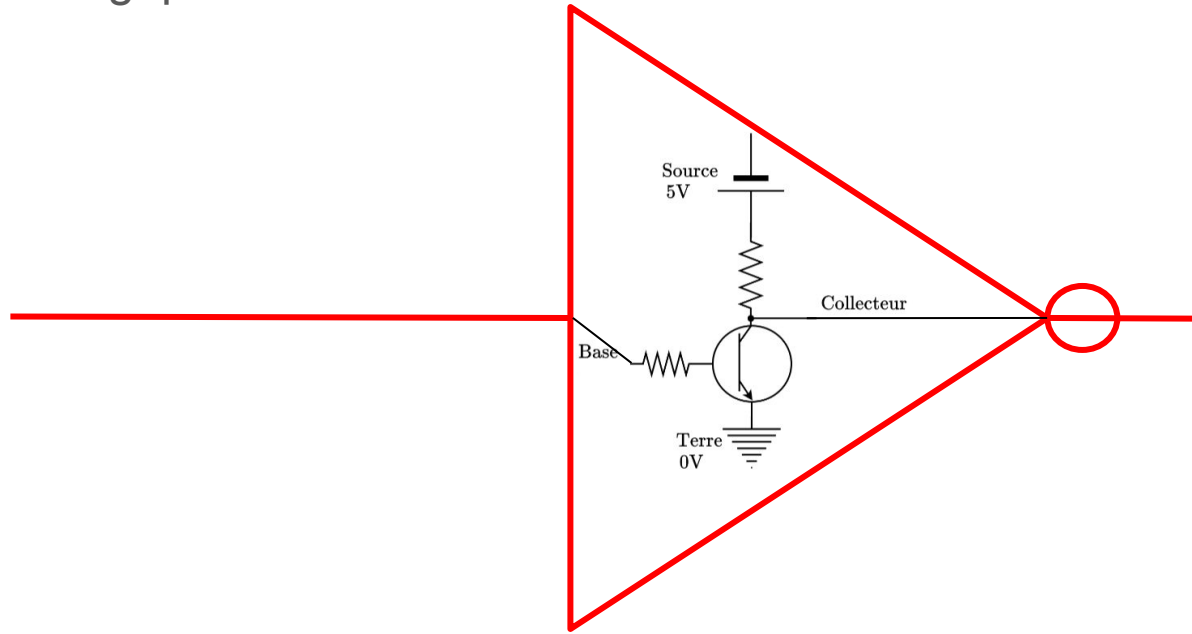
Fonction logique NOT :

Base **active** le transistor.
⇒ Le courant s'établit.
⇒ Chute de tension.



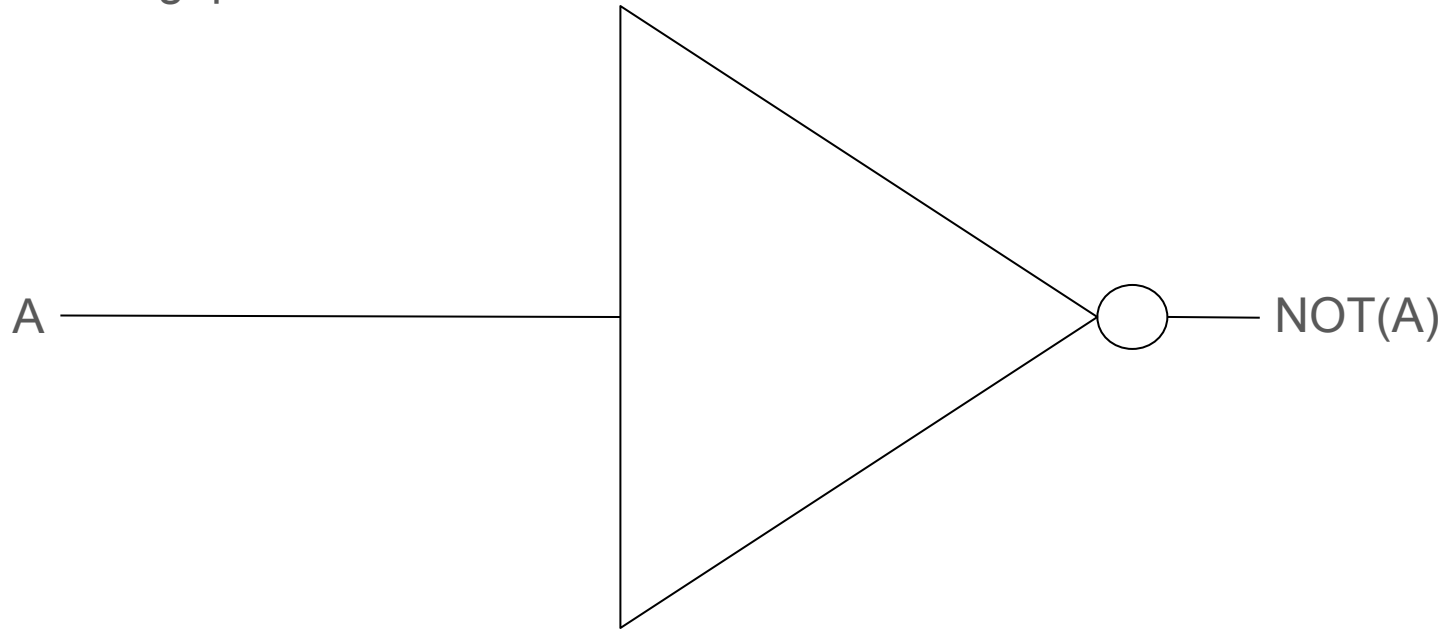
De transistors à portes logiques | **Abstraction**

Fonction logique NOT :



De transistors à portes logiques | **Abstraction**

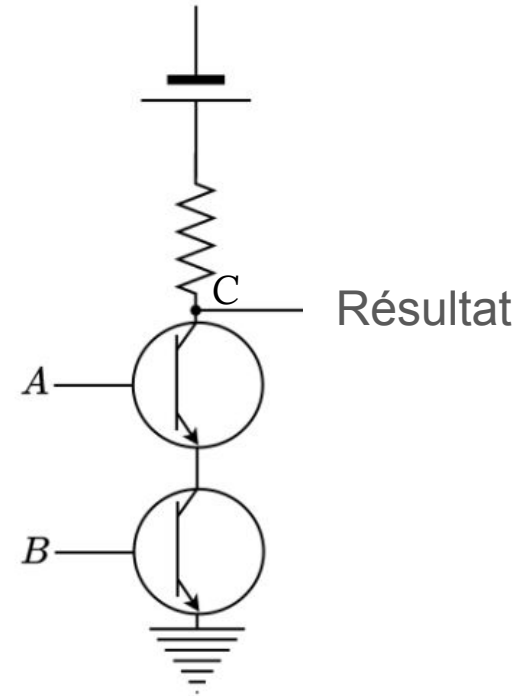
Fonction logique NOT :



Une première porte : transistors en série

- Il y a chute de tension au collecteur si le courant s'établit à travers l'un ET l'autre des transistors.
- Autrement dit : $U_c > 0$ seulement si au moins un des deux transistors n'est pas activé.

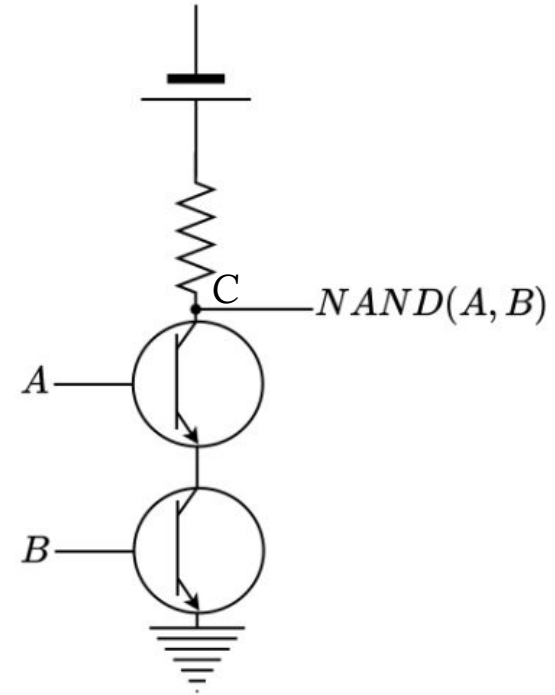
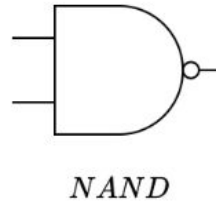
A	B	Résultat
haut	haut	
haut	bas	
bas	haut	
bas	bas	



Porte NAND

- Il y a chute de tension au collecteur si le courant s'établit à travers l'un ET l'autre des transistors.
- Autrement dit : $U_c > 0$ seulement si au moins un des deux transistors n'est pas activé.

A	B	$NAND(A, B)$
0	0	1
0	1	1
1	0	1
1	1	0

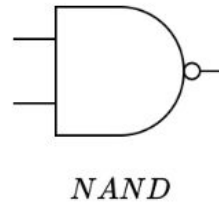


Porte NAND

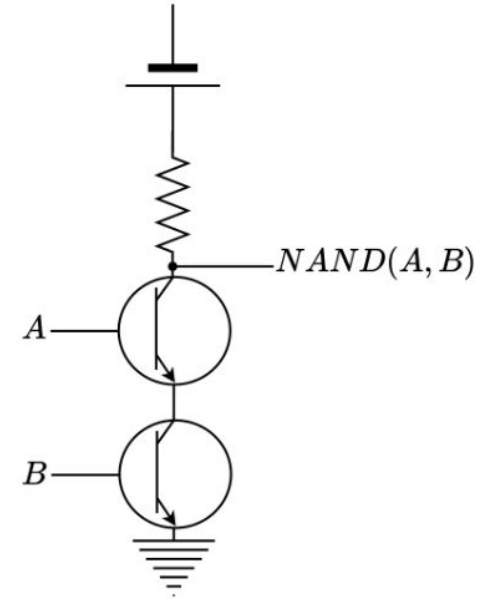
Fonction logique

A	B	$NAND(A, B)$
0	0	1
0	1	1
1	0	1
1	1	0

Porte logique



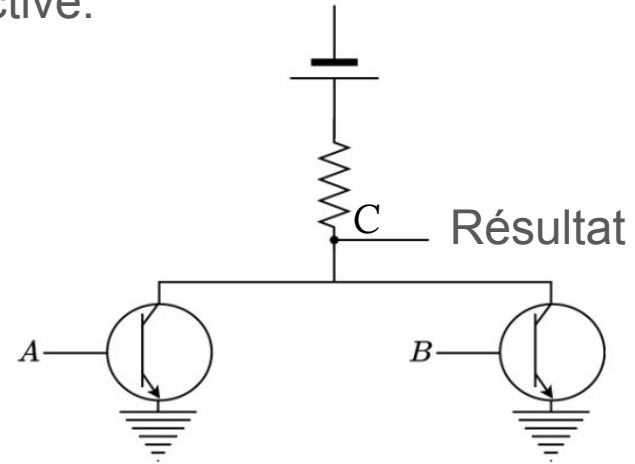
Circuit électronique



Une seconde porte : transistors en parallèle

- Il y a chute de tension dès que A OU B est activé.
- $U_c > 0$ seulement si aucun transistor n'est activé.

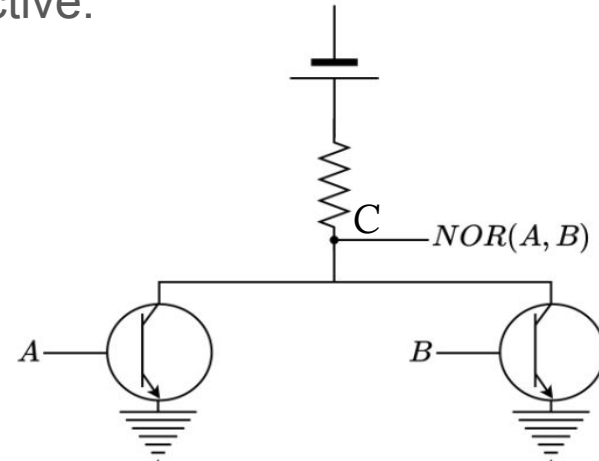
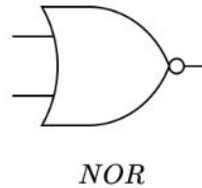
A	B	Résultat



Porte NOR

- Il y a chute de tension dès que A OU B est activé.
- $U_c > 0$ seulement si aucun transistor n'est activé.

A	B	$NOR(A, B)$
0	0	1
0	1	0
1	0	0
1	1	0



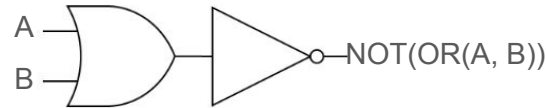
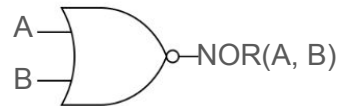
Combinaison de fonctions logiques

NAND est une inversion de AND au même titre que NOR est une inversion de OR.

A	B	AND	NAND	OR	NOR
0	0				
0	1				
1	0				
1	1				

Combinaison de fonctions/portes logiques

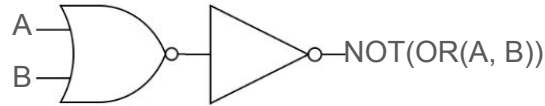
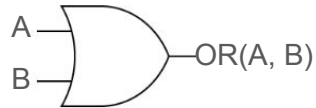
Exemple pour $\text{NOR}(A, B) = \text{NOT}(\text{OR}(A, B))$



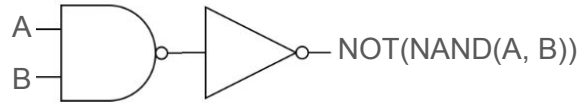
A	B	OR	NOR
0	0	0	1=not(0)
0	1	1	0=not(1)
1	0	1	0=not(1)
1	1	1	0=not(1)

Combinaison de fonctions/portes logiques

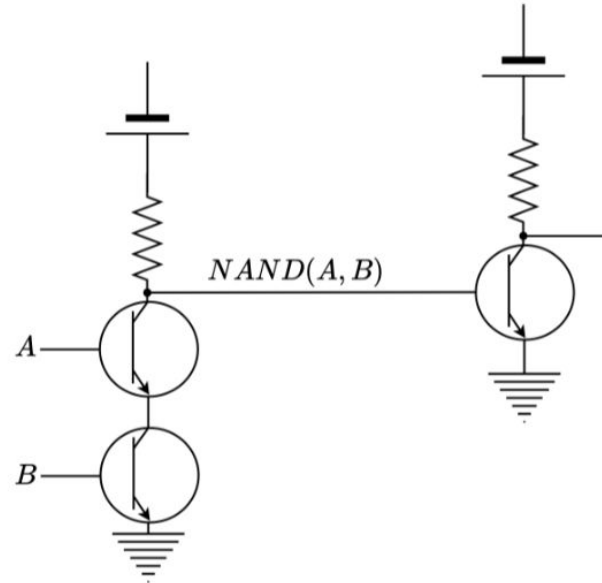
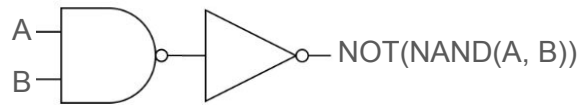
À l'inverse $OR(A, B) = NOT(NOR(A, B))$



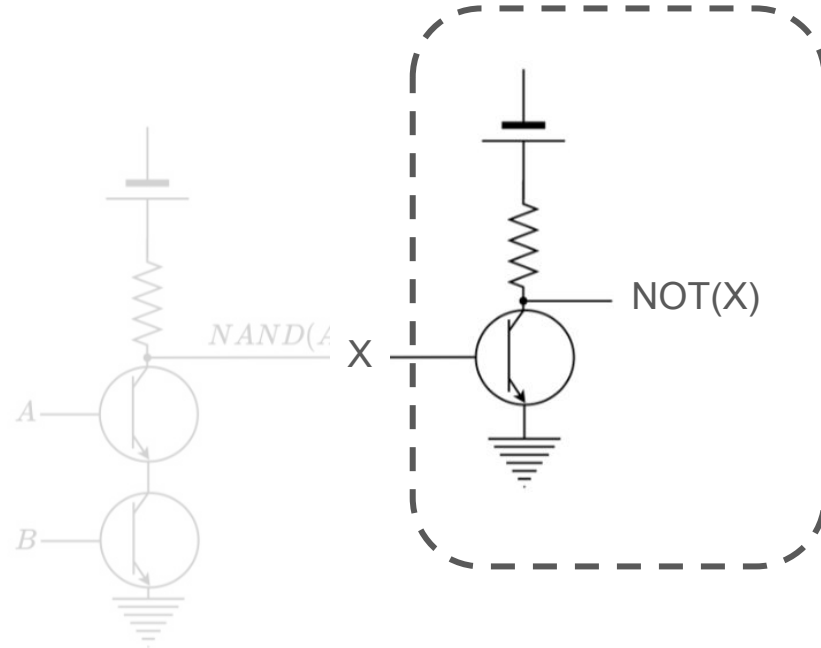
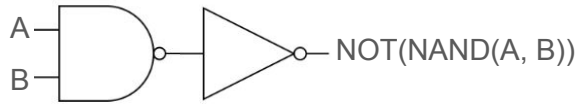
Ou encore $AND(A, B) = NOT(NAND(A, B))$



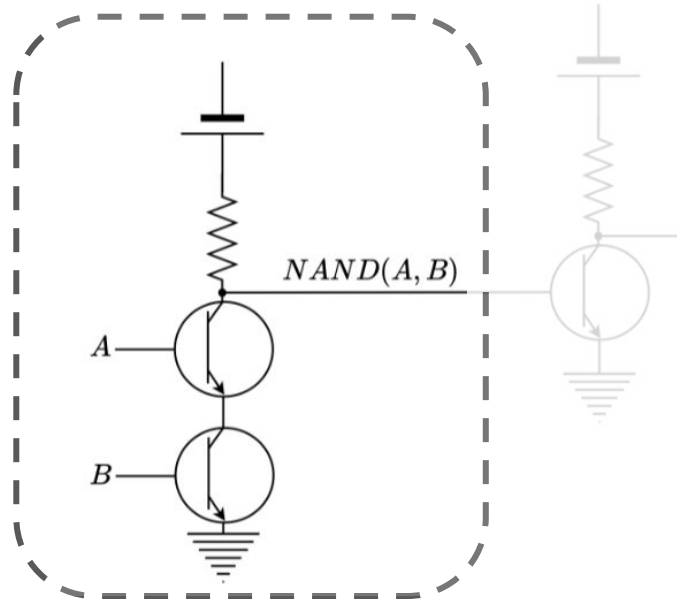
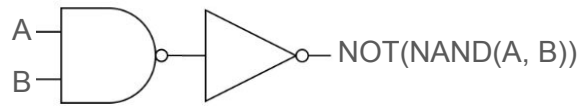
Exemple de porte AND à base de transistors



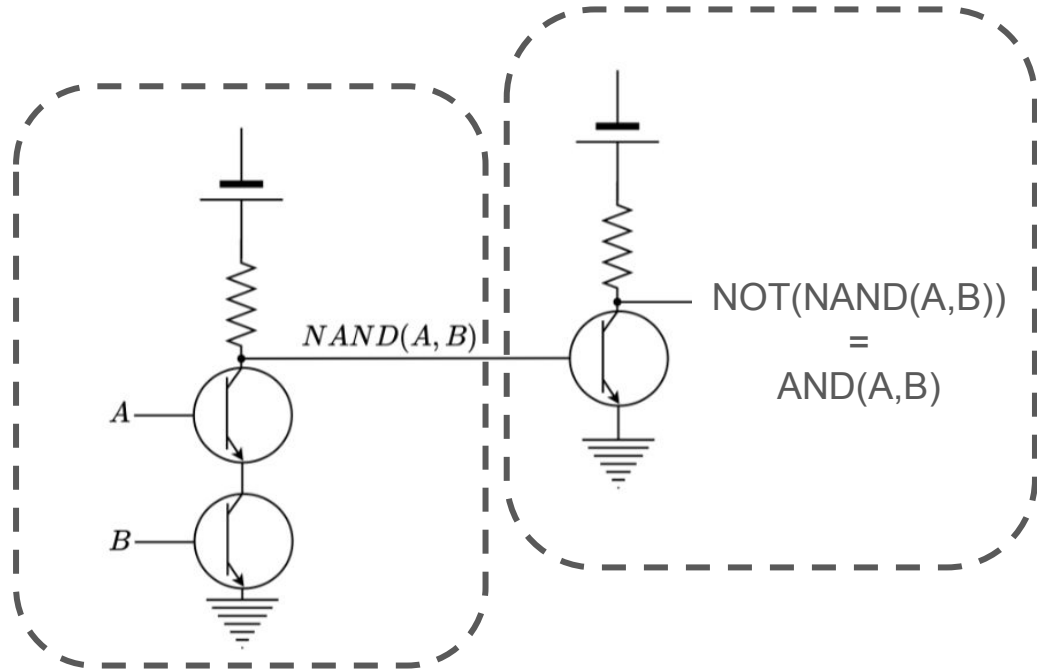
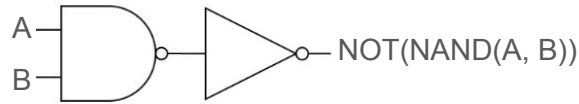
Exemple de porte AND à base de transistors



Exemple de porte AND à base de transistors



Exemple de porte AND à base de transistors

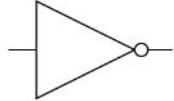
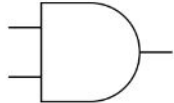
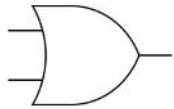


Algèbre de Boole

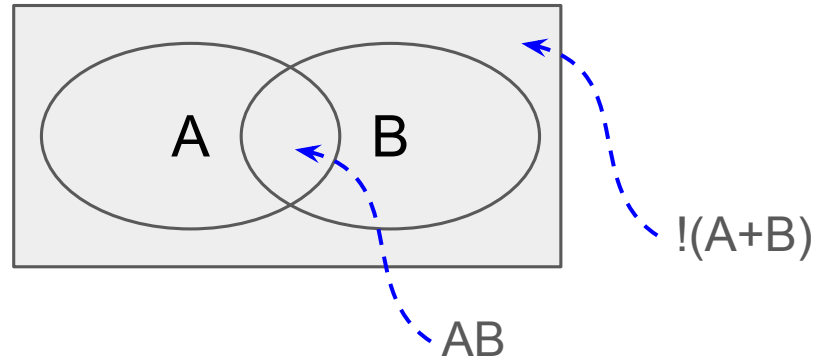
- Nous avons vu qu'un raisonnement logique pouvait être exprimé :
 - Comme une phrase du langage courant (ambigu).
 - Comme une table de vérité.
 - Comme un schéma (circuit logique). Cette dernière méthode peut s'implémenter comme un circuit électronique dans le monde physique, par exemple grâce à des transistors.
- Afin de simplifier et prouver l'équivalence de raisonnements logiques, nous allons maintenant utiliser une **notation et des règles mathématiques** : l'algèbre de Boole.

Fonctions logiques | **Notation**

1 = vrai et 0 = faux

Fonction	Algèbre	Logique	Programmation	Diagrammes logiques
NOT	!	$\neg$ ou $\sim$	!	
AND	.	$\wedge$	&& ou <i>and</i>	
OR	+	$\vee$	ou <i>or</i>	

Fonctions logiques | **Vision ensembliste**





Logique booléenne | **Exemples d'écriture**

Logique booléenne | Relations générales

- Comme on l'a vu, on peut dégager certaines relations générales.
Par exemple : $OR(A, B) = NOT(NOR(A, B))$
Ou encore (équivalent) : $A + B = !(!(A + B))$
- L'algèbre de Boole (vers 1850) formalise le calcul de propositions binaires.
Permet de transformer des phrases (intuitionnistes) logiques en expressions algébriques.
- Nécessaire car :
 - Tous les raisonnements ne sont pas aussi simples que celui de l'excès de vitesse.
 - **Simplification** d'expressions en apparence complexes → impact sur le circuit correspondant !
 - On veut pouvoir fournir des preuves formelles.

Logique booléenne | **Priorité des opérations**

- NOT(**être hors localité**) AND (**rouler à + de 50 km/h**) s'écrit : $\neg A \cdot B$


La négation est prioritaire sur les autres fonctions logiques.

- Pour le reste, on suit l'ordre des opérations habituel en mathématiques :
AND est prioritaire sur OR.

Logique booléenne | Règles - Double négation

- Pour l'exemple, revenons sur la correspondance obtenue par table de vérité :

$$\text{OR}(A, B) = \text{NOT}(\text{NOR}(A, B))$$

$$\text{C'est-à-dire : } A + B = \neg(\neg(A + B))$$

- À l'évidence, $X = \neg(\neg X)$
- Analogue à la double négation dans la langue française :
"ce n'est pas irréaliste" signifie "c'est réaliste".
"ce n'est pas faux" signifie "c'est vrai".

Logique booléenne | Preuve des règles

- Une façon de procéder pour les cas simples : vérification exhaustive.
- Exemple pour $\neg(x + y) = \neg x \neg y$

x	y	$\neg(x + y)$	$\neg x \neg y$
0	0		
0	1		
1	0		
1	1		

Logique booléenne | Règles importantes

Nom	Version <i>AND</i>	Version <i>OR</i>
Valeur nulle	$0 \cdot x = 0$	$1 + x = 1$
Valeur neutre	$1 \cdot x = x$	$0 + x = x$
Complément	$x!x = 0$	$x+!x = 1$
Commutativité	$xy = yx$	$x + y = y + x$
Associativité	$x(yz) = (xy)z$	$x + (y + z) = (x + y) + z$
Distributivité	$x(y + z) = xy + xz$	$x + yz = (x + y) \cdot (x + z)$
Idempotence	$xx = x$	$x + x = x$
Absorption	$x(x + y) = x$	$x + (xy) = x$
Théorème de De Morgan	$!(xy) = !x+!y$	$!(x + y) = !x!y$

Exemples de simplification

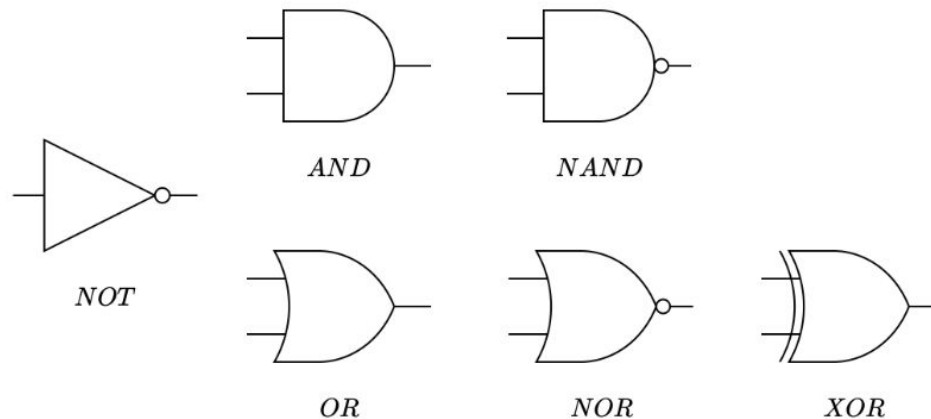
1) $xy + x = ?$ (sans utiliser absorption)

2) $C \cdot !(B + C) = ?$

Nom	Version <i>AND</i>	Version <i>OR</i>
Valeur nulle	$0 \cdot x = 0$	$1 + x = 1$
Valeur neutre	$1 \cdot x = x$	$0 + x = x$
Complément	$x!x = 0$	$x+!x = 1$
Commutativité	$xy = yx$	$x + y = y + x$
Associativité	$x(yz) = (xy)z$	$x + (y + z) = (x + y) + z$
Distributivité	$x(y + z) = xy + xz$	$x + yz = (x + y) \cdot (x + z)$
Idempotence	$xx = x$	$x + x = x$
Absorption	$x(x + y) = x$	$x + (xy) = x$
Théorème de De Morgan	$!(xy) = !x+!y$	$!(x + y) = !x!y$

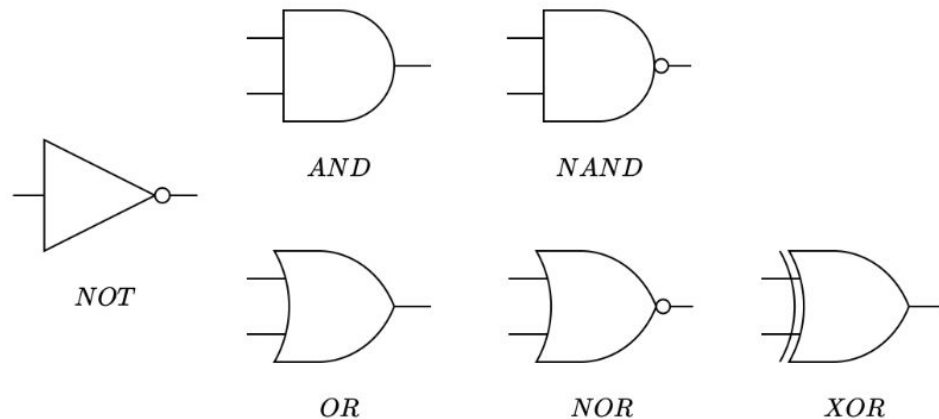
Logique booléenne | **Fonction logique universelle**

- Existe-t-il une fonction qui permette à **elle-seule** de formuler **toutes les autres** ?



Logique booléenne | Fonction logique universelle

- Existe-t-il une fonction qui permette à **elle-seule** de formuler **toutes les autres** ?
- Réponse : oui. Il en existe même deux.



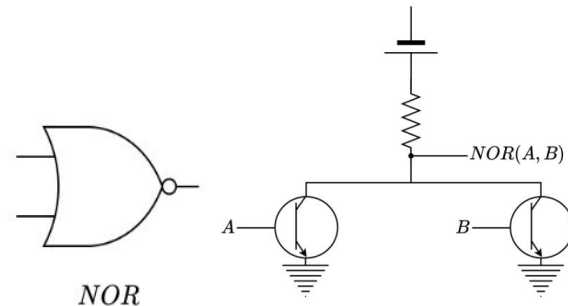
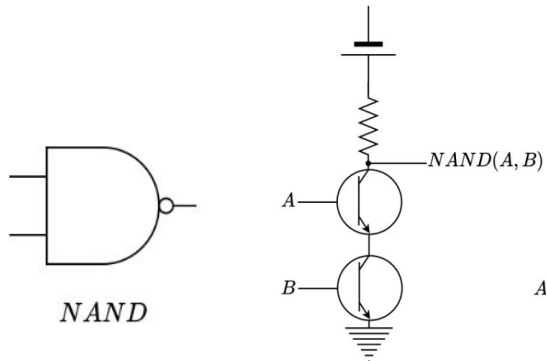


Logique booléenne | **Fonction logique universelle**

- Partons des trois fonctions qui nous semblent fondamentales : NOT, AND, OR, car on a déjà vu que le XOR ($A \oplus B = A!B + B!A$) et les couples AND-NAND et OR-NOR peuvent s'exprimer à l'aide de leur négation.
- Voyons si l'on peut réduire ce triplet de fonctions.
Arbitrairement, commençons par examiner OR:

Logique booléenne | Fonctions logiques universelles

- NOR est universelle au même titre que NAND (démonstration similaire).
- Il se trouve que ce sont justement les portes les plus faciles à construire avec des transistors.



Un exemple non trivial : nombres premiers à 3 bits

- Objectif : obtenir la table de vérité correspondant à la fonction logique $P(x,y,z)$ telle que $P(x,y,z) = 1$ si et seulement si le nombre " zxy "₂ est premier.
- Exemples :
 - $P(1,0,0) = 0$ car $100_2 = 4$ n'est pas premier.
 - $P(1,0,1) = 1$ car $101_2 = 5$ est premier.

Un exemple non trivial : nombres premiers à 3 bits

Objectif : obtenir la table de vérité correspondant à la fonction logique $P(x,y,z)$ telle que $P(x,y,z) = 1$ si et seulement si le nombre $(zxy)_2$ est premier.

Inputs			Output
z	y	x	P

Un exemple non trivial : nombres premiers à 3 bits

Objectif : obtenir la table de vérité correspondant à la fonction logique $P(x,y,z)$ telle que $P(x,y,z) = 1$ si et seulement si le nombre $(zxy)_2$ est premier.

z	y	x	P

Un exemple non trivial : nombres premiers à 3 bits

Listons les conditions pour avoir $P = 1$:

$$P = !zy!x + !zyx + z!yx + zyx$$

Simplifions :

Un exemple non trivial : nombres premiers à 3 bits

Listons les conditions pour avoir $P = 1$:

$$P = !zy!x + !zyx + z!yx + zyx = !zy + zx$$

Ce qui correspond au circuit :

Une approche méthodique pour l'expression d'une table

- Précédemment (nombre premier à 3 bits), nous avons listé toutes les façons de produire la valeur 1 en sortie, puis avons simplifié l'expression.
- Ces différentes façons de produire 1 sont combinées avec des OU logiques.
- Méthode : $\text{output} = \text{expression}_1 + \text{expression}_2 + \dots + \text{expression}_n$.
- Cette méthode se nomme la méthode des **mintermes**.

Mintermes et maxtermes

- Mintermes : lister toutes les façons de produire la valeur 1 en sortie (combinées avec des OU).
- Maxtermes : lister toutes les façons de produire la valeur 0 en sortie (combinées avec des OU), puis en effectuer la négation.

Exemple pour les maxtermes | **Nombre premier à 3 bits**

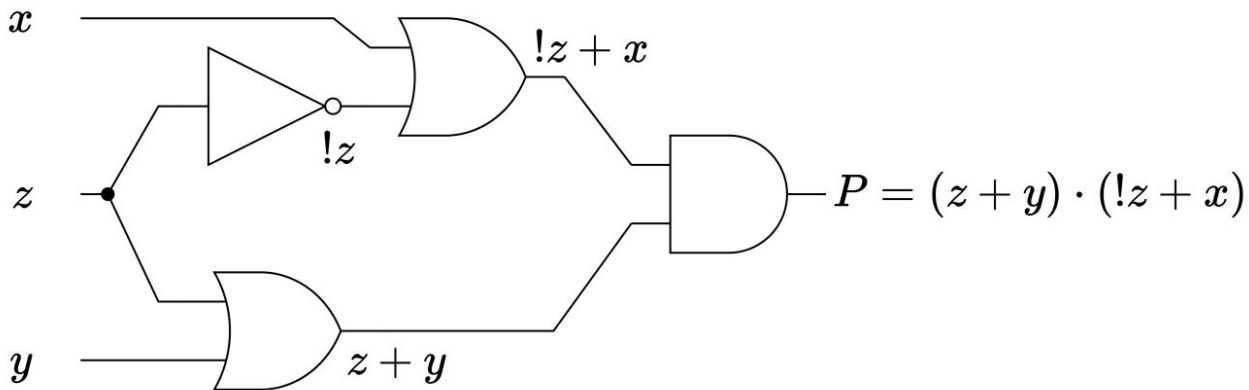
- Les entrées qui donnent $P = 0$ sont 000, 001, 100 et 110.
- Cela correspond à l'expression $\neg P = \neg z \neg y \neg x + \neg z \neg y x + z \neg y \neg x + z y \neg x$

Exemple pour les maxtermes | **Nombre premier à 3 bits**

- Les entrées qui donnent $P = 0$ sont 000, 001, 100 et 110.
- Cela correspond à l'expression $\neg P = \neg z \neg y \neg x + \neg z \neg y x + z \neg y \neg x + z y \neg x$
- Simplification : $\neg P = \neg z \neg y (\neg x + x) + z \neg x (\neg y + y) = \neg z \neg y + z \neg x$
- Négation : $P = \neg \neg P = \neg (\neg z \neg y + z \neg x) = \neg (\neg z \neg y) \cdot \neg (z \neg x) = (\neg \neg z + \neg \neg y) \cdot (\neg z + \neg \neg x)$
 $= (z + y) \cdot (\neg z + x)$

Exemple pour les maxtermes | Nombre premier à 3 bits

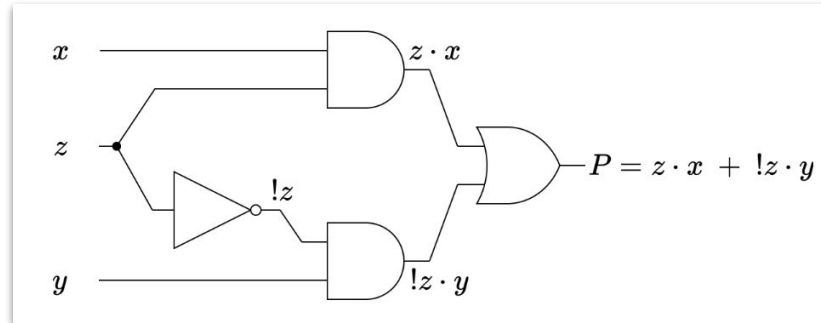
Finalement $P = (z + y) \cdot (!z + x)$



Note sur l'équivalence des deux derniers circuits

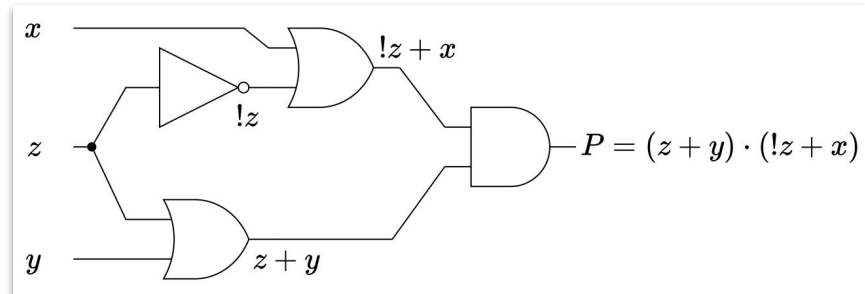
Circuit obtenu avec les mintermes :

$$P_m = zx + !zy$$



Circuit obtenu avec les maxtermes :

$$P_M = (z + y) \cdot (!z + x)$$



Note sur l'équivalence des deux derniers circuits

- Si nous n'avons pas fait d'erreur : $P_m = P_M$
- $P_M = (z + y) \cdot (!z + x) = z!z + zx + !zy + xy = zx + !zy + \mathbf{xy} = P_m + \mathbf{xy}$

Y a-t-il une erreur ?

Note sur l'équivalence des deux derniers circuits

- $zx + !zy + xy = zx + !zy$ correspond au théorème du consensus, ou théorème de la redondance.
- Preuve par l'algèbre de Boole en exercice.