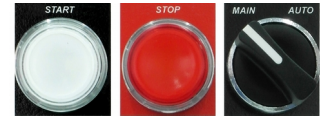


Introduction

■ Parmi les **informations** gérées par la fonction « traiter » d'un système, une partie est de nature dite **logique** – ou **binnaire**, ou encore **tout ou rien** (TOR) – i.e. avec seulement **deux valeurs possibles**, se matérialisant dans les circuits par deux niveaux de tensions distincts. On peut ainsi caractériser l'état d'un bouton (actionné ou non), d'une vanne (fermée ou ouverte), d'un moteur (en marche ou à l'arrêt), etc.



En sciences de l'ingénieur, au lieu d'employer le formalisme de la **logique classique** (notions de vrai/faux, etc.), on modélise ces aspects par des **variables booléennes**¹, qui prennent leur valeur dans l'ensemble des nombres réduit à **{0, 1}**. Ces variables peuvent être **élémentaires** – on dit aussi **atomiques**, au sens d'**indivisibles** – ou **composées**, i.e. **fonction** d'une ou plusieurs variables élémentaires, et on les appelle des **fonctions logiques**.

Pour chacune des fonctions logiques mises en œuvre par la fonction « traiter » d'un système, il faut identifier les **variables d'entrée** et de **sortie**, ainsi que d'éventuelles **variables internes** caractérisant divers **états** de la fonction « traiter » :

- on dit qu'une fonction opère en **logique combinatoire** si les valeurs prises par les variables de sorties ne dépendent que de la **combinaison des valeurs des variables d'entrée**, autrement dit si la fonction n'a pas de variables internes ni de bouclage d'une sortie sur une entrée ;
- dans le cas contraire, on parle de **logique séquentielle** – domaine abordé dans le chapitre suivant du cours.

■ Dans la pratique, les fonctions logiques sont implémentées de diverses manières selon l'**énergie** et la **technologie** de la partie commande du système. Encore très fréquent dans les années 1980, le recours à l'énergie pneumatique est maintenant exceptionnel et réservé à des milieux très contraignants (e.g. atmosphères explosives) où l'énergie électrique n'est pas utilisable. De plus, les **fonctions logiques câblées**, qui se justifient encore pour des systèmes simples, tendent à être remplacée par des **techniques microprogrammées**.

1. Éléments d'algèbre de Boole pour la logique

1.1 Généralités

■ L'**algèbre de Boole** peut être vue comme une **formalisation algébrique** de la **logique des propositions**² :

- aux **valeurs de vérité faux** et **vrai** correspondent respectivement les **valeurs booléennes 0** et **1** ;
- aux **connecteurs logiques non** (\neg), **et** (\wedge), **ou** (\vee)³ correspondent respectivement les **opérateurs booléens** :
 - le **complément à 1**, noté $\bar{}$, lu « barre » ou « non » : $\bar{a} = 1 - a$; on a donc $\bar{0} = 1 - 0 = 1$ et $\bar{1} = 1 - 1 = 0$;
 - le **produit booléen**, noté \cdot , lu « et »⁴ : $a \cdot b = a \times b$; on a donc $0 \cdot 0 = 0$, $0 \cdot 1 = 0$, $1 \cdot 0 = 0$ et $1 \cdot 1 = 1$;
 - la **somme booléenne**, notée abusivement $+$, lue « ou » : $0 + 0 = 0$ et $0 + 1 = 1 + 0 = 1 + 1 = 1$; la somme booléenne n'est donc **pas analogue à l'addition usuelle**⁵, car en **base 2 cyclique**, on a $1 + 1 = 0$! Il est donc vivement déconseillé de lire son symbole « plus », sous peine de confusions fâcheuses.

■ L'intérêt principal de l'algèbre de Boole est d'**unifier dans un même formalisme les aspects logiques, cardinaux et ordinaux** des données numériques codées dans un système, avec deux avantages notoires :

- **manipuler les expressions logiques** avec des règles proches de celles du **calcul usuel** (à quelques différences qui ne doivent pas être sous-estimées, liées notamment à la définition de la somme booléenne $+$) ;
- **effectuer des opérations logiques** (complément, etc.) **bit à bit** sur des **données numériques**, et des opérations algébriques (addition, etc.) sur des données logiques regroupées en nombres (entiers).

1. Du nom de George BOOLE, mathématicien britannique du XIX^e siècle, qui en est à l'origine.

2. Une **proposition logique** est un énoncé admis comme ne pouvant prendre que les valeurs **vrai** ou **faux**, e. g. « Socrate est un homme ». Formalisée dès le IV^e s. avant J. C. par l'école stoïcienne, la **logique des propositions** est l'étude des lois régissant ces éléments du discours.

3. Symbole \vee inspiré de l'initiale du mot latin *vel* qui signifie « ou » ; le symbole \wedge du connecteur « et » est choisi par « symétrie ».

4. Symbole souvent omis à l'usage (et donc non prononcé) dans les expressions booléennes, e.g. $a \cdot b = a \times b$ (comme la multiplication).

5. C'est pourquoi la somme booléenne était initialement notée par le symbole $\dot{+}$ qui n'a pas survécu aux contraintes typographiques. C'est aussi pourquoi elle n'est pas définie en intention par une formule algébrique, mais seulement en extension par valeurs.

■ Par construction, toute expression de la logique des propositions trouve son équivalent en algèbre de Boole et réciproquement. Les **lois** – ou **égalités remarquables** – énoncées ci-dessous sont à connaître :

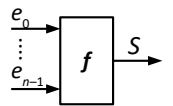
• commutativité	$a \cdot b = b \cdot a$	$a + b = b + a$
• associativité	$a \cdot (b \cdot c) = (a \cdot b) \cdot c = a \cdot b \cdot c$	$a + (b + c) = (a + b) + c = a + b + c$
• distributivité	$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$	$a + (b \cdot c) = (a + b) \cdot (a + c)$
• idempotence	$a \cdot a = a$	$a + a = a$
• élément neutre	$a \cdot 1 = a$	$a + 0 = a$
• élément absorbant	$a \cdot 0 = 0$	$a + 1 = 1$
• complémentarité	$a \cdot \bar{a} = 0$	$a + \bar{a} = 1$
• absorption	$a \cdot (a + b) = a$	$a + (a \cdot b) = a$
	$a \cdot (\bar{a} + b) = a \cdot b$	$a + (\bar{a} \cdot b) = a + b$
• lois de De Morgan⁶	$\overline{a \cdot b} = \bar{a} + \bar{b}$	$\overline{a + b} = \bar{a} \cdot \bar{b}$
• involution	$\overline{\bar{a}} = a$	

Par **substitutions inductives**, la plupart de ces lois peuvent être généralisées à k occurrences de variables. Par exemple, en adoptant les notations usuelles de l'algèbre, les lois d'idempotence s'écrivent $a^k = a$ et $ka = a$.

1.2 Fonctions et expressions booléennes

■ Comme sur tout ensemble E , on a sur $\{0, 1\}$ la notion d'**endomorphisme** (fonction de la forme $f: E \rightarrow E$) mais, comme $\text{card } \{0, 1\} = 2$, il n'existe que quatre fonctions distinctes de la forme $\{0, 1\} \rightarrow \{0, 1\}$. La notion pertinente est celle de **fonction de n variables**, i.e. de la forme $f: \{0, 1\}^n \rightarrow \{0, 1\}$ où $\{0, 1\}^n$ est l'**ensemble produit** itéré n fois de $\{0, 1\}$, qui est de **cardinal 2^n** . On appelle fonction booléenne ou logique ce type de fonctions.

Une fonction logique f peut aussi être vue comme une fonction d'une **variable entière e** dont chacun des n bits e_k (pour $k = 0$ à $n - 1$) est une des variables booléennes de f . Elle se schématise par un **bloc fonctionnel** à n entrées e_0, e_1, \dots, e_{n-1} et une **sortie $S = f(e_0, e_1, \dots, e_{n-1})$** – cf. ci-contre⁷.



■ À l'instar de toute fonction, une fonction logique f se définit par son **graphe⁸**, et ce d'autant plus aisément que le cardinal des ensembles de départ et d'arrivée de f sont finis. Mieux qu'un nuage de points sur un diagramme, on présente ce graphe sous la forme d'une **table de vérité** donnant la valeur **0** ou **1** de $S = f(e_0, e_1, \dots, e_{n-1})$ pour toutes les combinaisons des valeurs de ses variables, présentées en n colonnes et 2^n lignes. La fonction est ainsi définie **en extension** par la $n + 1^{\text{e}}$ colonne de la table (cf. ci-contre).

e_{n-1}	...	e_1	e_0	S
0	...	0	0	0/1
0	...	0	1	0/1
...
1	...	1	1	0/1

Comme en algèbre, une fonction logique f peut être **indéfinie** en certains points :

- soit parce que la combinaison des valeurs d'entrée ne se produit **jamais** : on note – la « non valeur » de f ;
- soit parce que la valeur de f est **indistinctement 0 ou 1 (astabilité)** : on note alors \emptyset ou X sa valeur.

■ En algèbre, on montre qu'il existe $(\text{card } F)^{(\text{card } E)}$ **fonctions distinctes** de la forme $f: E \rightarrow F$.

De la forme $f: \{0, 1\}^n \rightarrow \{0, 1\}$, on peut donc définir 2^{2^n} **fonctions logiques**.

- Pour $n = 1$, (une seule variable d'entrée, notée a), on a **4 fonctions** :

« **oui** » ($f_1 = a$), « **non** » ($f_2 = \bar{a}$), et les 2 fonctions constantes $f_0 = 1$ et $f_3 = 0$ (cf. ci-contre).

a	f_0	f_1	f_2	f_3
0	0	0	1	1
1	0	1	0	1

- Pour $n = 2$ (deux entrées a et b), on a **16 fonctions**, dont « et », « ou », et d'autres opérateurs remarquables :

□ la **fonction de Sheffer**, appelée aussi opérateur « on » (en anglais nand, i.e. « non et ») : $f_{14} = a \uparrow b = \overline{a \cdot b}$

□ la **fonction de Pierce**, appelée aussi opérateur « ni » (en anglais nor, i.e. « non ou ») : $f_8 = a \downarrow b = \overline{a + b}$

□ l'**opérateur « ou exclusif »** (en anglais xor pour **exclusive or**) : $f_6 = a \oplus b = \overline{a \cdot b} + \overline{\bar{a} \cdot \bar{b}}$.

6. Du nom d'Auguste DE MORGAN, mathématicien britannique contemporain de BOOLE.

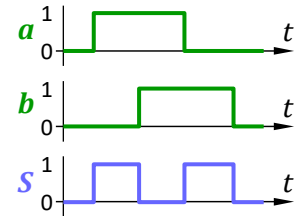
7. On note souvent la sortie d'une fonction logique par la lettre Q , similaire à la lettre O (pour « output ») délaissée car trop semblable à 0.

8. Rappel : le graphe d'une fonction $f: E \rightarrow F$ est le sous-ensemble Γ de $E \times F$ tel que $(x, y) \in \Gamma \Leftrightarrow y = f(x)$ (cf. cours, chap. 1.3).

Les 16 fonctions logiques à 2 variables sont toutes définies dans la table de vérité ci-dessous :

b	a	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
symbole		.						\oplus	+	\downarrow	\odot					\uparrow	

■ Une autre façon de représenter le graphe d'une fonction logique est le chronogramme. Avec $n + 1$ échelles de temps synchrones, on trace l'un en dessous de l'autre le chronogramme de chaque entrée e_k et de la sortie S de la fonction, en choisissant pour les entrées des créneaux qui couvrent toutes les combinaisons possibles des valeurs. À la lecture du chronogramme de S , on reconstitue la table de vérité de la fonction. Par exemple, l'opérateur ou exclusif $a \oplus b$ est défini par le chronogramme ci-contre : $S = 1$ si et seulement si $a = 0$ et $b = 1$ ou $a = 1$ et $b = 0$.



Mais dès trois variables d'entrée, le chronogramme devient une représentation fastidieuse pour exprimer toutes les combinaisons possibles des valeurs. La table de vérité est donc la méthode privilégiée.

■ À partir de quatre variables d'entrée, la table de vérité comporte au moins 16 lignes et devient également malcommode. On préfère alors procéder *en intention* par la formation d'une expression booléenne composée de fonctions déjà définies (notamment les opérateurs booléens) appliquées à des variables ou à des constantes booléennes. Par exemple, la fonction « majorité » est définie par l'expression : $maj(a, b, c) = ab + bc + ca$.

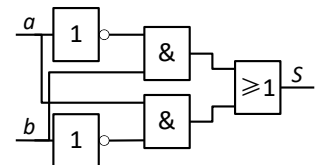
Comme usuellement en algèbre, les expressions booléennes supposent des règles de priorité entre opérateurs (dans l'ordre : complément, produit, somme) et peut employer des parenthèses pour exprimer une autre hiérarchie de composition. On peut aussi appliquer le principe de substitution pour remplacer uniformément toutes les occurrences d'une variable par une expression composée. Par convention, on écrit en minuscule les variables atomiques (ne pouvant se décomposer, car représentant un aspect élémentaire du système, e.g. l'état d'un bouton, d'un capteur TOR, etc.) et en majuscule les variables composées. (sortie d'une fonction, etc.).

■ Une autre façon de représenter l'expression algébrique d'une fonction logique f est le logigramme, formalisme qui révèle la structure arborescente de sa composition :

- chaque instance d'un opérateur ou d'une sous-fonction employé dans la composition de f est représentée par son bloc fonctionnel (cf. le tableau des opérateurs, p. 8) ; elle constitue un nœud, i.e un embranchement ;
- chaque occurrence d'une variable d'entrée de f est représentée par une liaison pendante à gauche ; c'est une feuille de l'arbre ; toutes les occurrences d'une même variable sont reliées entre elles ;
- la sortie S de la fonction, est représentée par une liaison pendante à droite ; c'est la racine de l'arbre.

Ci-contre est représenté le logigramme de l'opérateur ou exclusif ($a \oplus b = a\bar{b} + \bar{a}b$) ; les blocs « 1 », « & », « ≥ 1 » symbolisent respectivement les opérateurs non, et, ou.

Le logigramme est une représentation pertinente comme première étape pour l'implémentation câblée d'une fonction logique définie algébriquement.



1.3 Obtention et réduction des expressions booléennes

■ Il est fréquent qu'une fonction logique ne soit initialement connue que par les conditions pour laquelle elle est vraie (c'est le principe d'énoncé d'un cahier des charges). Cela revient en fait à spécifier les combinaisons des valeurs des variables d'entrée (i.e. les lignes de la table de vérité) pour lesquelles la fonction vaut 1. Implicitement, pour toutes les autres lignes, la fonction vaut 0, ce qui détermine complètement la table.

On détermine alors une expression de la fonction comme somme des produits des atomes (si la variable vaut 1) ou de leur complément (si la variable vaut 0) pour laquelle la fonction vaut 1. Par exemple, pour la fonction f_9 (correspondant à l'équivalence logique) définie dans la table (cf. supra), on a $f_9 = a \odot b = \bar{a}b + a\bar{b}$.

■ L'expression d'une fonction logique obtenue comme somme de produits peut ensuite être réduite (i.e. simplifiée), soit par calcul algébrique, soit par la méthode des tableaux de Karnaugh, soit par la procédure algorithmique de Quine & Mac Cluskey. Ces aspects sont traités dans des ouvrages spécialisés.

II. Implémentation des fonctions logiques

II.1 Généralités

Dans les systèmes techniques, une fonction logique mise en œuvre pour le traitement de données peut être implémentée de deux façons fondamentalement différentes :

- soit en **logique câblée**, i.e. à l'aide de composants implémentant eux-même des opérateurs booléens ou des fonctions plus complexes, connectés entre eux selon un schéma qui s'apparente au logigramme de la fonction, qu'on appelle usuellement un **circuit** ; limitée par des contraintes matérielles (existence des composants, encombrement, difficultés de modifications ultérieures, etc.), cette technologie est **de moins en moins utilisée** mais reste privilégiée pour sa **fiabilité** sur des fonctions de sécurité (arrêt d'urgence, modes manuels dégradés, etc.) ou dans des contextes particuliers (absence d'énergie électrique) ;
- soit en **logique programmée**, i.e. sous formes d'instructions programmées dans une unité de traitement à **microcontrôleur**, **microprocesseur** ou **automate programmable** ; sans limites de complexité, facilement **modifiable**, contrôlable à distance, permettant la structuration des données, cette technologie est **de loin la plus employée** et rend souvent inutiles certaines tâches préalables d'analyse comme la réduction algébrique, dont la finalité n'est parfois que la simplicité de câblage.

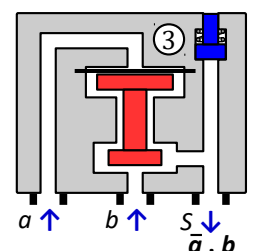
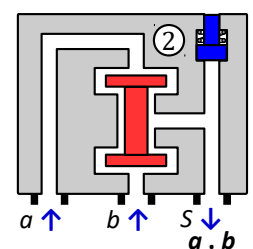
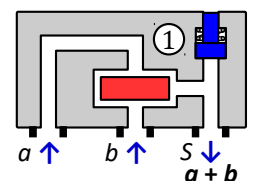
II.2 Circuits pneumatiques

■ L'**énergie pneumatique** employée dans les systèmes est véhiculée par de l'**air comprimé** à une pression d'environ **6 bars**, qui sert principalement de **source de puissance** pour des actionneurs comme les vérins pneumatiques. Elle n'est presque plus employée par la **chaîne d'information**, sauf dans des contextes où l'électricité n'est pas utilisable : atmosphères explosives, voisinage de champs magnétiques intenses...

■ Dans un **système de commande pneumatique**, les opérateurs booléens sont implémentés par des **cellules** constituées en **boîtiers étanches** munis d'**orifices de connexion** de **tuyaux souples** pour les entrées et la sortie. Leur fonctionnement repose sur les **positions d'un clapet** (ci-dessous en **rouge**) soumis aux flux d'air comprimé présents ou non aux entrées. Souvent, un petit piston monostable⁹ (ci-dessous en **bleu**) indique par sa position (rentré/sorti) l'**état de la sortie** (0/1).



- La **cellule « ou »** (cf. fig. 1) est une cellule **passive** (i.e. autonome en énergie) où le clapet permet la mise sous pression de la sortie ($S = 1$) si au moins l'une des deux entrées **a** ou **b** est sous pression (si une seule entrée est sous pression, l'autre est obturée par le clapet pour que l'air entrant soit évacué uniquement vers **S**) ; dans le cas contraire, s'il n'y a aucune entrée en pression ($a = b = 0$), il n'y a pas de pression en sortie.
- La **cellule « et »** (cf. fig. 2) est une cellule **passive** où le clapet permet la mise sous pression de la sortie ($S = 1$) uniquement en position d'équilibre des pressions venant des deux entrées **a** et **b** ; sinon ($S = 0$), la sortie est soit obstruée (la pression sur une seule entrée bloque le passage vers **S**), soit non alimentée (aucune pression nulle part).
- La **cellule « inhibition de b par a »** (cf. fig. 3) est une cellule **passive** où le clapet est surdimensionné du côté de l'entrée **a** et actionné via une membrane souple ; si **a** est sous pression ($a = 1$), le clapet obture la sortie **S** même si l'entrée **b** est sous pression (pas d'équilibre) ; sinon (si $a = 0$), la sortie **S** est alimentée par la pression de l'entrée **b**, donc $S = 0$ si $b = 0$ et $S = 1$ si $b = 1$; en définitive, $S = \bar{a} \cdot b$.
- La **cellule « non »** est une cellule **active** (i.e. non autonome) réalisée par une cellule **inhibition** dont l'entrée **b** est reliée à une **alimentation** ($b = 1$) : on a donc $S = a \cdot 1 = \bar{a}$.
- La **cellule « oui »** est une cellule **active** ayant une conception similaire à une cellule **et**, mais dont l'entrée **a** est membranée et dont l'entrée **b** est reliée à une alimentation, i.e. $b = 1$: on a alors $S = a \cdot 1 = a$; une cellule **oui** sert à régénérer la pression d'un signal logique affaibli par des pertes.



■ Cette technologie est donc très **encombrante** et **rudimentaire** – même s'il existe d'autres cellules (mémoire, temporisation) dont le principe sera abordée au chapitre suivant. Globalement, l'implémentation d'une fonction logique composée en technologie pneumatique correspond exactement à son **logigramme**.

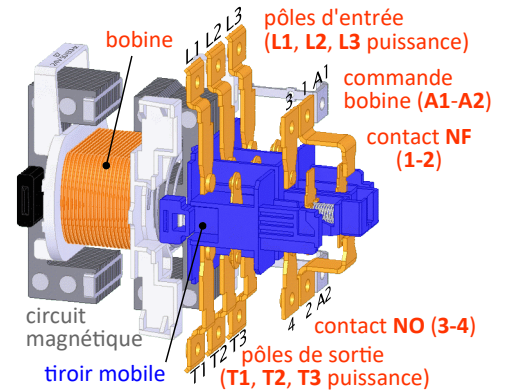
9. Un composant est dit *monostable* s'il revient spontanément (souvent via un ressort) en position de repos lorsqu'il n'est pas actionné.

II.3 Circuits électrotechniques

■ Les **machines industrielles** et les équipements stationnaires de grandes dimensions (e.g. pont roulant, ascenseur...) sont pourvus d'une **armoie électrique** qui commande l'alimentation en énergie électrique du système. Outre les appareils de sectionnement et de protection (disjoncteurs, coupe-circuits, relais thermiques), on y trouve des **contacteurs de puissance** (cf. exemple en photo ci-contre) qui assurent la jonction entre les circuits de puissance et les circuits de commande, et des **contacteurs auxiliaires pour implémenter des fonctions logiques**.



■ Un **contacteur de puissance** (cf. vue interne ci-contre) est un appareil de commutation d'un circuit électrique principal commandé par un circuit électrique secondaire via un **électro-aimant** et sa **bobine**. Si la bobine est excitée par la mise sous tension de ses bornes **A1-A2**, le contacteur se « ferme » : son tiroir mobile recule et raccorde les **pôles d'entrée** aux **pôles de sortie**, qui forment ses **contacts principaux**. Si la bobine n'est plus excitée, un **ressort de rappel** remet le tiroir mobile dans sa position au repos (appareil monostable).

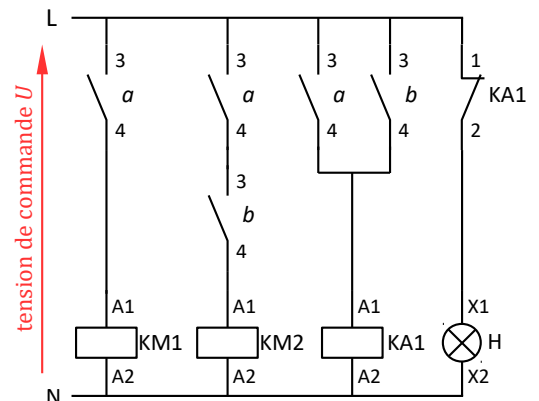


Un contacteur de puissance possède aussi des **contacts auxiliaires, normalement ouverts (NO – bornes repérées 3-4)** et **normalement fermés (NF – bornes repérées 1-2)**¹⁰ qui donnent respectivement l'image directe et inverse de son **état** ouvert ou fermé.

Un **contacteur auxiliaire** est un contacteur qui n'a que des contacts auxiliaires et pas de contacts de puissance. Il sert à implémenter une **mémoire intermédiaire** (1 bit) pour réaliser des fonctions logiques plus complexes.

■ Dans un système électrotechnique, les fonctions logiques ne sont pas implémentées par des « cellules » comme en technologie pneumatique, mais à partir de **contacts électriques** (i.e. des interrupteurs qui donnent l'image de l'état de composants : contacteurs, boutons, capteurs, relais...). Ces contacts constituent les **entrées** du système. Les **sorties** sont soit des **bobines** de contacteurs principaux qui commandent directement le circuit de puissance (départ moteur, etc.), soit des composants de signalisation (voyants, etc.). Elles sont mises sous tension par un générateur de **tension de commande** (usuellement, **24 V~**) via des **contacts d'entrée**. Selon le type de ces contacts (**NO** ou **NF**) et leurs dispositions respectives, en **série** ou en **parallèle**, on peut réaliser toutes les fonctions logiques imaginables :

- un seul contact **NO** alimentant la bobine d'un contacteur réalise une **fonction « oui »** (e.g. ci-contre $KM1 = a$) ;
- deux contacts **NO** disposés **en série** réalisent une **fonction « et »** ; ils doivent être l'un et l'autre fermés pour que la bobine qu'ils alimentent soit sous tension (e.g. ci-contre $KM2 = a \cdot b$) ;
- deux contacts **NO** disposés **en parallèle** réalisent une **fonction « ou »** ; il suffit que l'un ou l'autre soit fermé pour que la bobine qu'ils alimentent soit sous tension (e.g. ci-contre $KA1 = a + b$) ;
- un seul contact **NF** alimentant la bobine d'un contacteur réalise une **fonction « non »** ; pour compléter une fonction composée f , on passe par un **contacteur auxiliaire** alimenté par f dont on emploie un contact **NF** (e.g. ci-contre $H = \overline{KA1} = \overline{a + b}$).



■ Outre les contacteurs auxiliaires pour implémenter des mémoires, il existe aussi des blocs de **contacts auxiliaires temporisés** qui permettent de déclencher des commandes retardées (e.g. la fermeture d'une porte quelques secondes après appui sur un bouton). Mais comme en technologie pneumatique, ces fonctionnalités restent assez rudimentaires et deviennent malcommodes à implémenter pour des fonctions logiques vraiment complexes. Les contraintes d'encombrement sont telles que lorsque la commande d'un système nécessite la mise en œuvre de fonctions logiques complexes, on recourt à un **automate programmable**. Certains modèles sont à peine plus gros qu'un contacteur et offrent bien plus de possibilités (cf. la photo ci-contre du module **Zelio Schneider Electric**, avec 8 entrées et 4 sorties, programmable en langages LD et FBD).

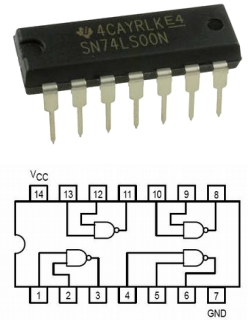


10. En anglais, le sigle NF s'écrit NC pour *normally closed* (quant au sigle NO, il coïncide en français avec l'anglais *normally open*).

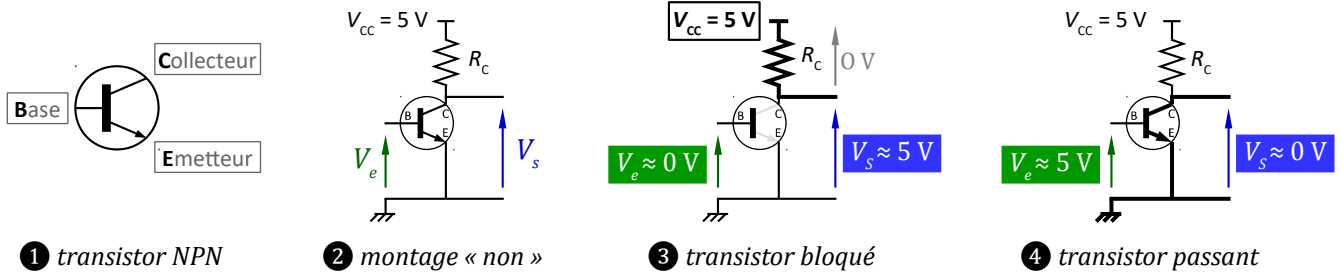
II.4 Circuits électroniques

■ En dehors de cas simples comme ceux vus précédemment, l'**électronique** est privilégiée pour réaliser la partie commande des systèmes techniques en raison de **multiples avantages** : complexité de traitement presque illimitée, compacité, très faible consommation électrique, adaptation à la production en grande série, faible coût, etc.

En technologie câblée, on implémente des fonctions logiques grâce à une vaste gamme de **circuits intégrés** (« puces ») conditionnés en boîtiers à broches soudables sur circuit imprimé¹¹. Tous les opérateurs booléens remarquables – qu'on appelle dans ce contexte des **portes logiques** – sont disponibles souvent en exemplaires multiples ou avec plus de deux entrées ; cf. ci-contre le circuit intégré SN7400 à 4 portes *nand* dont les symboles sont représentés en norme ANSI (*American national standards institute*).



■ La technologie TTL (*transistor transistor logic*) repose sur le **transistor en commutation** détaillé ci-après.



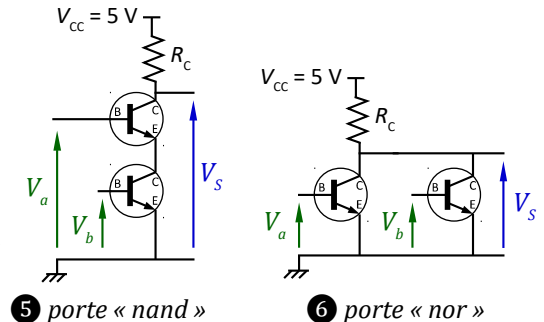
Le **transistor**¹² dit **bipolaire** à jonctions semi-conductrices NPN – cf. fig ① – est un composant à trois broches appelées respectivement **base**, **collecteur** et **émetteur**. Le montage le plus élémentaire ② consiste à alimenter le collecteur en tension continue $V_{cc} = 5\text{ V}$ via une résistance R_c et relier l'émetteur à la masse ; le transistor est commandé par la **tension d'entrée** V_e appliquée sur la base et délivre au collecteur une **tension de sortie** V_s .

- Si $V_e \approx 0\text{ V}$, le transistor est dans **l'état bloqué** ③ : la liaison collecteur-émetteur n'est pas conductrice ; donc le collecteur est au potentiel $V_s \approx 5\text{ V}$ (il n'y a pas de chute de tension dans R_c car aucun courant n'y circule).
- Si $V_e \approx 5\text{ V}$, le transistor est dans **l'état passant** (on dit aussi **saturé**) ④ : la liaison collecteur-émetteur devient conductrice ; donc le collecteur est relié à la masse, et on a $V_s \approx 0\text{ V}$.

Le montage simpliste ② réalise donc une porte « non » puisqu'en logique booléenne, on obtient $S = \bar{e}$.

■ Si maintenant on monte deux transistors **en série** comme sur la figure ⑤ ci-contre, en considérant comme entrée les tensions de base V_a et V_b et comme sortie la tension du collecteur amont V_s , il faut que les deux transistors soient passants, autrement dit que $V_a = V_b \approx 0\text{ V}$, pour avoir $V_s \approx 0\text{ V}$; dans tous les autres cas, on a $V_s \approx 5\text{ V}$. Le montage, simpliste également, constitue donc une porte « nand ».

Si enfin on monte deux transistors **en parallèle** comme sur la figure ⑥, on obtient $V_s \approx 5\text{ V}$ uniquement si $V_a = V_b \approx 0\text{ V}$; dans tous les autres cas, $V_s \approx 0\text{ V}$. Le montage constitue donc une porte « nor ».



Par l'algèbre de Boole, on montre aisément que les opérateurs « **nand** » « **nor** » sont **universels** : l'un comme l'autre suffisent à engendrer toutes les fonctions booléennes par compositions et liaisons des entrées. Mais pour optimiser les temps de réponse, la consommation électrique et la fabrication, les portes logiques TTL commercialisées par les fabricants sont implémentées par des montages plus subtils (avec *pôle totem*, etc.)¹³.

■ Grâce à leur **miniaturisation extrême**, les transistors sont intégrables en grands nombres (plus de $10^6/\text{mm}^2$) pour former des composants très complexes. Comme circuits logiques combinatoires, on trouve notamment :

- les **multiplexeurs**, qui partagent une seule voie pour plusieurs signaux (les **démultiplexeurs** font l'inverse),
- les **opérateurs arithmétiques** (additionneurs, soustracteurs, etc.) de nombres entiers codés en binaire,
- les **comparateurs**, qui testent l'égalité entre deux mots binaires,
- les **transcodeurs**, qui convertissent divers formats de mots binaires (e.g. binaire naturel → Gray).

11. Il existe aussi des conditionnements plus petits dits CMS (*composants montés en surface*, en anglais *SMD surface mounted device*).

12. Acronyme anglais de *transfer resistor* ; appellation choisie par les *Bell Laboratories* en 1948.

13. Et la technologie CMOS, qui se passe complètement de résistors, fait appel à d'autres montages (*pull up & pull down networks*).

II.5 Programmes micro-informatiques

■ En **technologie micro-programmée**, l'implémentation des fonctions logiques d'un système dépend de son **environnement de programmation**. Il s'agit d'un **logiciel** qui permet notamment :

- de composer le **programme** de commande du système dans un ou plusieurs **langages** dits de **haut niveau** (i.e. d'un usage commode pour le cerveau humain) ;
- de traduire ce programme en **instructions** de **bas niveau** (i.e. exécutables par une machine) et de les **implanter en mémoire** (on dit aussi *charger* voire *décharger*) dans le système.

La technologie diffère selon qu'on emploie soit une **carte électronique à microcontrôleur**, comme pour les systèmes embarqués et les équipements de grande série (ascenseurs, etc.), soit un **automate programmable industriel (API)**, comme pour les machines et équipements de production (convoyeurs, etc.).

■ Pour les API, il existe **5 langages** de programmation normalisés CEI 61131-3, tous compilés¹⁴ :

- **ST** (de l'anglais **structured text**) est un langage **littéral** de **haut niveau** et **structuré**. Sa syntaxe, proche de celle de langages comme Pascal¹⁵ ou Delphi, est explicite (mots-clefs intelligibles comme **begin**, **end**, **if**, **repeat**, etc.). Il est **le plus utilisé** pour les programmes complexes car il est le plus puissant des cinq langages en terme d'expressivité. Apprécié des informaticiens, il facilite l'application des méthodes du génie logiciel dans le contexte de l'informatique industrielle.
- **IL** (de l'anglais **instructions list**) est conçu comme un langage **assembleur** (de **bas niveau**) pour les techniciens familiers de ce type de programmation ; il est **en désuétude** car très laborieux à écrire et lire.
- **LD** (de l'anglais **ladder diagram**, le mot *ladder* signifiant « échelle ») est un langage **semi-graphique** conçu pour les électriciens : il exprime les fonctions logiques sous forme de **schémas électriques** avec des **contacts** et des **bobines**. Mais pour l'emploi de variables numériques, il nécessite des extensions en langage ST ;
- **FBD** (de l'anglais **functional bloc diagram**) est un langage **semi-graphique** conçu pour les automaticiens : il exprime les fonctions booléennes sous forme de **logigrammes**. Mais comme pour le langage LD, l'emploi de variables numériques et l'expression de fonctions numériques nécessitent des extensions en langage ST.
- **SFC** (de l'anglais **sequential function chart**) est un langage **semi-graphique** qui implémente l'outil de spécification français *Grafcet*. Conçu pour les automaticiens, il permet d'exprimer les **changements d'état du système** sous forme d'un **diagramme étapes-transitions**. Les fonctions logiques exprimant les conditions externes d'évolution sont programmées dans d'autres langages (ST, LD ou FBD).

■ Pour les **cartes électroniques**, la programmation s'effectue surtout avec des **langages littéraux**, notamment :

- **C**, **C++**¹⁶ ou leur dérivés (pour la plupart des cartes, dont *Arduino*), qui sont des langages **compilés** de haut niveau, à syntaxe semi-symbolique (« { » pour **begin**...), pouvant aussi manipuler directement la mémoire ;
- **Python** (pour des cartes comme *Raspberry Pi*), un langage **interprété**¹⁷ de **haut niveau**, à syntaxe explicite mais épurée (l'indentation des instructions participe à définir leur structuration), nativement orienté objet.

■ En langage littéral, les fonctions logiques sont implémentées par leur **expression booléenne**, en respectant les **spécificités syntaxique** du langage utilisé :

- les **opérateurs booléens** sont en général écrits par leur **nom anglais** (**and**, **or**, **not**, **xor**, **nand**, **nor**) – ou parfois un symbole (e.g. **&** pour **et**, **!** pour **not**, etc.) ; ce sont des **mots-clés** du langage ;
- les **variables** sont désignées chacune par un **identificateur** choisi par le programmeur de façon suffisamment explicite, mais concise, pour aider à comprendre son rôle dans le programme ; avec l'ajout de commentaires, cette bonne pratique améliore grandement la lisibilité du code ;
- les **valeurs de vérité** constantes sont désignées numériquement (**0**, **1**) ou littéralement (**true**, **false**) ;
- le **parenthésage** vient en complément des règles de priorité des opérateurs définies par le langage.

On peut également effectuer des opérations booléennes sur des **mots mémoires** ou des **variables entières** de même échelle : elles s'effectuent **bit à bit** sur le code binaire des valeurs de ces variables.

14. Le programme est entièrement analysé et traduit en instructions machine avant de pouvoir être exécuté.

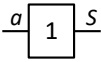

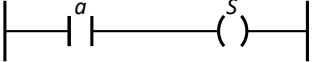
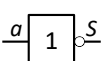
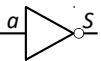
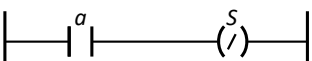


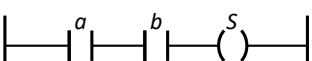
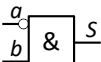

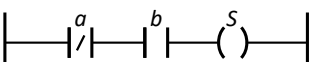
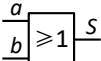


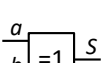

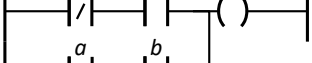
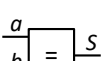

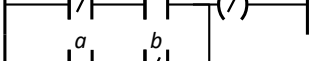
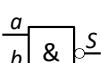

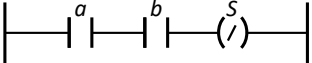
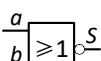


15. Langage inventé vers 1970 pour l'*apprentissage* de la programmation structurée, mais qui a aussi des applications industrielles.

16. Inventé dans les années 1970 pour réécrire le système d'exploitation *Unix*, le langage de programmation *C* est l'un des plus employés au monde. Il permet de coder des opérations complexes de façon plus synthétique qu'en Pascal, mais souvent au prix d'une moindre lisibilité ; *C++* est une extension *orientée objet* de *C*, i.e. permettant de définir des *classes* d'objet avec la notion d'*héritage*.

17. Les instructions du programme sont traduites en langage machine au fur et à mesure de l'exécution du programme.

III. Tableau récapitulatif des opérateurs booléens

Hormis « inhibition », tous les opérateurs sont à connaître ou à retrouver rapidement à partir des autres.

noms	table de vérité		symbole normalisé *		mot clé ST / C	implémentation en langage LD **
			CEI 60617-12	ANSI/IEEE 91-1984		
« oui » fonction identité	a 0 1	$S = a$ 0 1		 buffer	-	
« non » complément, négation	a 0 1	$S = \bar{a}$ 1 0		 inverter	not !	
« et » produit booléen, conjonction	b a 0 0 0 1 1 0 1 1	$S = a \cdot b$ 0 0 0 1			and &&	
inhibition	b a 0 0 0 1 1 0 1 1	$S = \bar{a} \cdot b$ 0 0 1 0			-	
« ou » somme booléenne, disjonction	b a 0 0 0 1 1 0 1 1	$S = a + b$ 0 1 1 1			or 	
« oux » disjonction exclusive	b a 0 0 0 1 1 0 1 1	$S = a \oplus b$ 0 1 1 0			xor	 $S = \bar{a} \cdot b + a \cdot \bar{b}$
équivalence logique	b a 0 0 0 1 1 0 1 1	$S = a \odot b$ 1 0 0 1			xnor	 $S = \bar{a} \cdot b + a \cdot \bar{b} = a \cdot b + \bar{a} \cdot \bar{b}$
« non et » « on » fonction de Sheffer	b a 0 0 0 1 1 0 1 1	$S = a \uparrow b$ 1 1 1 0			nand	 $S = a \cdot b = \overline{\bar{a} + \bar{b}}$
« non ou » « ni » fonction de Pierce	b a 0 0 0 1 1 0 1 1	$S = a \downarrow b$ 1 0 0 0			nor	 $S = a + b = \overline{\bar{a} \cdot \bar{b}}$

* Le cercle symbolisant la négation est parfois remplacé par un triangle : \blacktriangle

** Les symboles | | et | / | représentent un contact, respectivement de type NO et NF ; les symboles () et (/) une bobine de type NO et NF.