Jean-Philippe Rennard 11/2000

LogiCell 1.0

On a vu dans l'introduction aux AC que le Jeu de la Vie permet de construire une machine universelle de Turing. LogiCell montre comment l'utilisation de canons comme générateurs, de planeurs comme signaux et de mangeurs comme sorties, permet de traiter l'ensemble des fonctions booléennes.

L'applet permet de tester théoriquement n'importe quelle équation booléenne à quatre entrées (celles-ci pouvant être répétées indéfiniment).

Un ensemble d'applications combinatoires montre l'usage qui peut être fait de ces fonctions.

Enfin et comme il s'agit d'un Conway pur, je n'ai pas pu résister à la tentation de tester certaines figures classiques.

Démarrage rapide

Au démarrage, il faut sélectionner un mode de fonctionnement, soit :

1. ***Équation*** : Dans ce mode, vous pouvez tester une équation booléenne théoriquement sans limitation. Pour entrer une nouvelle équation, sélectionnez le bouton "Nouv." et saisissez l'équation à partir des différents boutons d'entrée. La validation par "OK" n'est disponible que si l'entrée est syntaxiquement complète. Fixez ensuite les valeurs des entrées booléennes ABCD et lancez l'algorithme. Une sortie Noire signifie Faux, une blanche Vrai.
2. ***Va-et-vient*** : Vous avez un interrupteur à chacune des extrémités de votre couloir, le changement d'état de l'un des deux interrupteurs inverse l'état de la lampe. C'est le seul mode dynamique, vous n'avez pas besoin de valider l'entrée, l'automate se lance automatiquement. L'équation est ~(AwB).
3. ***Additionneur binaire*** : C'est une calculatrice gérant la somme de deux chiffres d'un bit (on fait difficilement plus limité je le concède !). Les deux valeurs entrées sont A (faux = 0, vrai = 1) et B. Le résultat est naturellement en binaire et s'exprime sur deux bits. Par exemple si A et B sont vrais, la somme A+B est 1+1=2 soit en notation binaire 10b.
4. ***Afficheur*** : Les écrans cristaux liquides de base affichent les différents caractères sur la base d'une matrice de 7 segments. Si l'on représente un chiffre à partir des 4 entrées binaires DCBA (A étant le bit de poids faible, DCBA=0111b vaut 7 par exemple) l'allumage de chacun des segments peut être géré par une fonction booléenne. Choisissez un chiffre - la valeur décimale s'affiche dans la zone de texte - et regardez les différents segments s'allumer. Dans ce premier mode, 3 des 7 segments sont donnés afin d'accélérer le traitement.
5. ***Additionneur binaire sur 2 bits*** : C'est le même que le précédent si ce n'est qu'il

peut additionner deux nombres exprimés sur 2 bits DC+BA. Par exemple, avec D vrai, C faux, B faux et A vrai, on a la somme 10b + 01b, soit en décimal : 2 +1, le résultat binaire est 11b soit 3. Les valeurs décimales sont affichées dans la zone de texte.

6. ***Afficheur 2*** : C'est le même que l'afficheur précédent, mais il gère cette fois l'ensemble des 7 segments. Soyez patient...

Pour le reste, l'interface est suffisamment dépouillée pour se comprendre spontanément.

Fonctionnement de LogiCell

A- Brefs rappels sur l'algèbre de Boole [1]

L'algèbre de Boole a été introduite en 1847 afin de proposer une formulation algébrique des propositions logiques. Cette formulation algébrique est nécessaire du fait de l'ambiguïté et de la complexité des propositions exprimées en langage naturel. Considérons par exemple les propositions suivantes :

- Si la nuit, tous les chats sont gris et si Garfield est un chat alors, la nuit Garfield est gris.

On reconnaît un syllogisme. Le syllogisme est une expression logique composée de la *majeure*, qui par l'intermédiaire de la *mineure*, permet de déduire la *conclusion*. Les deux premiers éléments sont les *prémisses*. A l'origine (Vème Siècle avant JC) il s'agissait essentiellement d'une méthode dialectique permettant de codifier le discours. C'est Aristote qui l'a généralisé au domaine scientifique. Cette méthode est d'une grande efficacité ; elle permet de conclure à la validité de la construction précédente (du moins dans sa construction logique). Considérons maintenant la proposition suivante :

- Si la souris a été dévorée par un chat et si Garfield est un chat, alors Garfield a dévoré la souris.

Cette seconde proposition est naturellement fausse, mais cette erreur n'apparaît pas spontanément, il est nécessaire de mener un raisonnement (en l'occurrence : Garfield n'est pas le seul chat existant) pour en prendre conscience. Cet exemple est trivial, mais imaginez un ensemble complexe de propositions liées, il devient alors rapidement impossible de conclure à la validité de la conclusion. L'algèbre de Boole permet de résoudre ce type de problème.

Cette algèbre travaille sur deux valeurs : 1 et 0 (Vrai et Faux) et utilise trois opérateurs :

1. La conjonction (notée : ^) c'est à dire le ET logique ;
2. La disjonction (notée : v) c'est le OU logique ;
3. La négation (notée : ~) c'est à dire le NON logique.

Les propriétés de ces opérateurs peuvent s'exprimer dans une table de vérité.

Le ET logique (^)

p	q	p ^ q
1	1	1
1	0	0
0	1	0
0	0	0

Ici, une proposition est vraie seulement si les deux prémisses sont vraies : "S'il fait beau et si il y a de la neige alors j'irai au ski.", il suffit que l'une des deux conditions ne soit pas remplie pour que la proposition "j'irai au ski" soit fausse.

Le OU logique (v)

p	q	p v q
1	1	1
1	0	1
0	1	1
0	0	0

Ici, il suffit que l'une des prémisses soit vraie pour que la conclusion le soit : "Si mon réveil ne sonne pas ou si ma voiture refuse de démarrer alors je serai en retard au travail."

Le NON logique (~)

p	~p
1	0
0	1

Cet opérateur sert tout simplement à inverser la valeur d'une proposition.

Un autre opérateur est couramment utilisé (notamment dans LogiCell), il s'agit du ou exclusif (xor) noté w :

XOR (w)

p	q	pwq
1	1	0
1	0	1

0	1	1
0	0	0

A la différence du Ou classique, la validité conjointe des deux propositions induit une conclusion fausse. Cet opérateur n'est pas élémentaire, il peut être construit à partir des trois précédents, par exemple : p w q = (p^~q)v(~p^q). De manière générale, tous les opérateurs (implication, équivalence) peuvent être construits à partir des trois opérateurs élémentaires.

Les lois de composition des opérateurs permettent à l'algèbre de Boole de gérer de manière rigoureuse un ensemble complexe de propositions liées.

Cette algèbre est utilisée de manière massive aussi bien au sein des mathématiques (théorie de la démonstration, probabilités...) que dans un cadre beaucoup plus pratique (automatisme, informatique...). En ce qui nous concerne, son intérêt provient du fait que les trois opérateurs booléens sont en théorie nécessaires et suffisants pour construire une Machine Universelle de Turing[2].

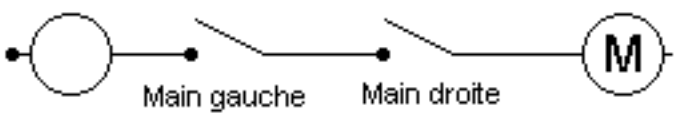
B- Automatisme et algèbre de Boole

Pour montrer l'usage pratique qui peut être fait des fonctions booléennes, nous avons choisi dans LogiCell de présenter quelques applications d'automatisme.

Un automatisme est un système placé entre un opérateur et une machine. L'opérateur fixe les entrées de l'automatisme en fonction du résultat voulu, et l'automatisme réalise un ensemble d'opérations qui déterminent la valeur des sorties envoyées à la machine. Les réflexions menées pour résoudre les problèmes d'automatisation ont conduit à l'élaboration d'un ensemble d'outils mathématiques ("automatique") dont l'utilisation a largement débordé le domaine d'origine.

Prenons l'exemple simple d'une double commande. Imaginez une presse à emboutir. Il s'agit d'une machine actionnant avec une pression importante un poinçon qui, en déformant un morceau de métal, va lui donner la forme désirée. Par exemple, vos tubes d'aspirine sont réalisés par la déformation d'un bout de métal ayant sensiblement la forme d'une pièce de monnaie. Pour s'assurer que les mains de l'opérateur ne sont pas dans la zone du poinçon, on l'oblige à actionner simultanément deux interrupteurs, un pour chaque main. Le schéma électrique est élémentaire :

Une double commande



Pour que l'énergie électrique puisse atteindre le moteur (ou en l'occurrence le relais

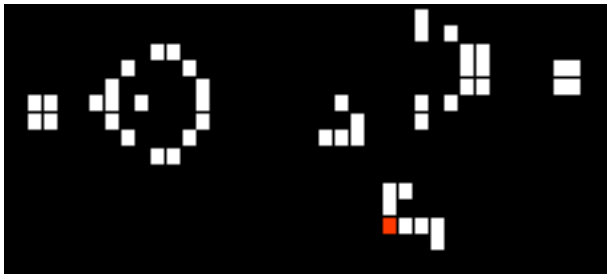
qui va déclencher le vérin hydraulique), les deux interrupteurs doivent être fermés. Deux conditions doivent être remplies simultanément : on reconnaît ici l'opérateur booléen ET. De la même manière, si l'on branche les deux interrupteurs non pas en série comme ici, mais en parallèle, on obtient la fonction OU. Il suffira que l'un des deux interrupteurs soit fermé pour que la sortie soit actionnée.

L'association des fonctions booléennes permet ainsi à travers un traitement complexe des entrées, de construire des "programmes" - des algorithmes - calculant les sorties souhaitées.

La sortie d'une double commande dépend exclusivement de l'état des entrées à l'instant *t*. On parle alors d'application combinatoire. Il est fréquent dans la pratique que l'on souhaite déterminer la sortie non seulement en fonction des entrées à l'instant *t*, mais aussi de certains états du système à l'instant *t-1*, ce sont les applications séquentielles[3]. Les "automates programmables" utilisés dans l'industrie disposent ainsi de fonctions de mémorisation et de temporisation. LogiCell est limité aux applications combinatoires, on verra toutefois que celles-ci peuvent être complexes.

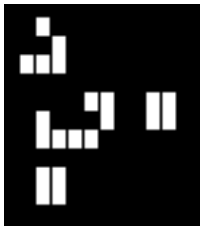
C- Principes de base

Logicell est un Conway pur. Les entrées sont générées par un canon de période 30 associé à un bloqueur.



L'entrée proprement dite se trouve sur la cellule rouge. Si elle est à vrai (ou 1) le bloqueur va disparaître et laisser le planeur se propager. Sinon évidemment le planeur est absorbé.

La sortie est gérée par un mangeur. La cellule affichée en rouge n'est activée que dans le cas où le mangeur absorbe un planeur. Cette cellule est la sortie.



Enfin, des canons simples (c'est à dire sans bloqueur) sont utilisés pour construire les

opérateurs logiques. Leur intérêt vient de la capacité des planeurs à s'annihiler mutuellement. Dans LogiCell, ce processus se fait en deux étapes, la première rencontre produit deux blocs (2x2) puis ceux-ci sont détruits par les planeurs suivants. Il aurait suffit d'introduire un décalage vertical de 1 cellule pour éviter la phase des blocs. La périodicité du canon le permettant, j'ai positionné les planeurs au même niveau pour simplifier la construction des opérateurs.

Les seules liaisons entre l'applet et l'automate sont donc :

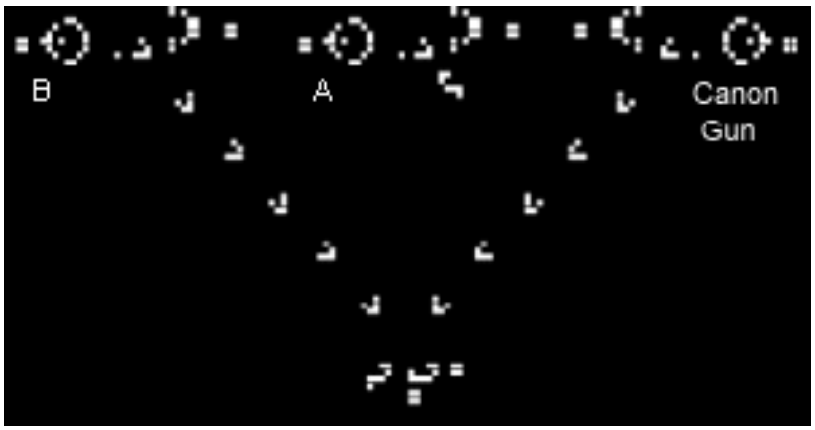
1. La position de la cellule d'entrée. On a naturellement un canon et donc une cellule d'entrée par entrée de l'équation (ABCD). Si une entrée apparaît plusieurs fois dans la même équation, elle sera représentée par autant de canons que d'occurrences.
2. La position de la cellule de sortie. On a une sortie unique par équation.

Strictement aucun autre traitement des entrées et sorties que celui de l'automate n'est utilisé. C'est bien l'automate et lui seul qui résoud le problème.

D- Construction des opérateurs booléens

Les trois portes logiques ET, OU, NON suffisent à gérer l'ensemble des opérateurs booléens (Non-Et ou Non-Ou auraient également pu convenir). Ces portes sont construites de la façon suivante :

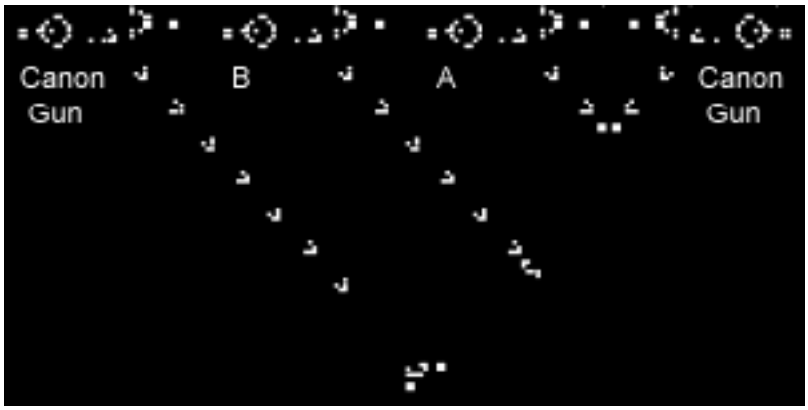
Porte ET avec B vrai et A faux



La porte est constituée de deux entrées et d'un canon de direction opposé. La sortie est sur la trajectoire du planeur de l'entrée B. Les configurations sont :

1. Les deux entrées sont fausses : aucun planeur ne rencontrant la sortie, le résultat est faux. On note que le bloqueur qui se trouve au même niveau que la sortie est utilisé dans ce cas pour éviter que le planeur du canon ne se propage hors de l'opérateur.
2. A est vrai et B est faux : le canon bloque le planeur de A. Même résultat.
3. A est faux et B est vrai : Le planeur de B est bloqué par le canon et ne peut donc rejoindre la sortie.
4. A et B sont vrais : Le canon est bloqué par le planeur de A, B peut activer la sortie

Porte OU avec A et B vrais



La porte OU est constituée de deux entrées, d'un canon de même direction et d'un canon de direction opposée. La sortie est sur la trajectoire du canon de gauche. Les configurations sont :

1. Les deux entrées sont fausses : Le canon de gauche est bloqué par le canon de droite.
2. A est vrai et B est faux : A bloque le canon de droite ; le canon de gauche peut rejoindre la sortie.
3. A est faux et B est vrai : B bloque le canon de droite ; le canon de gauche peut rejoindre la sortie.
4. A et B sont vrais : A bloque le canon de droite ; B est arrêté par le bloqueur ; le canon de gauche peut rejoindre la sortie.

Porte NON avec A faux

NON A (~A) renvoie le contraire de A, il est constitué d'une entrée et d'un canon de direction opposée. La sortie est sur la trajectoire du canon. Les configurations sont :

1. A est vrai : le planeur de A bloque le canon qui ne peut donc pas rejoindre la sortie ; le résultat est faux.

2. A est faux : le canon se propage librement vers la sortie ; le résultat est vrai.

On note que la porte A inverse la direction de la sortie. Si une entrée utilise un planeur se déplaçant vers la droite, la sortie est sur un planeur gauche et inversement.

Les autres opérateurs logiques peuvent être construits à partir des trois précédents. Par exemple, LogiCell construit l'opérateur XOR (ou exclusif) comme étant :

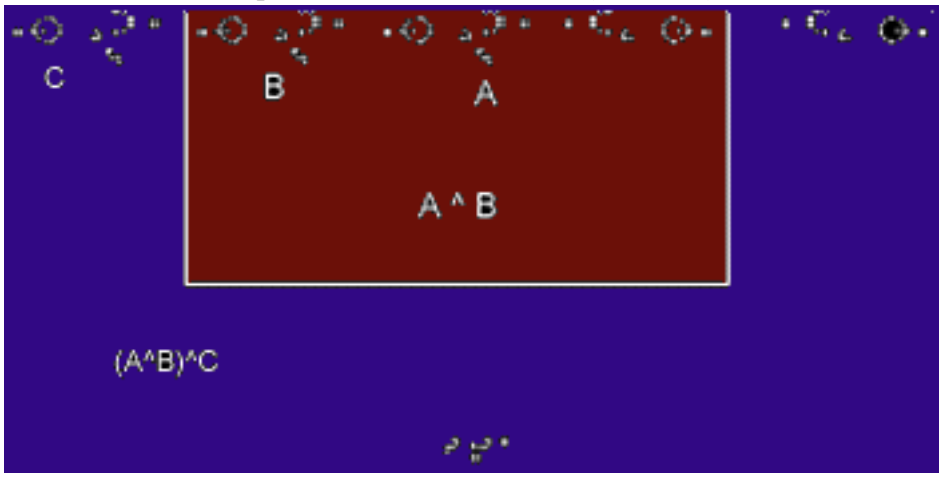
A Xor B = (A Ou B) Et Non(A Et B)
AwB = (AvB) ^ ~(A^B)

Pour mémoire le Ou Exclusif est vrai si une seule entrée est vraie.

E- Construction d'une équation

Une équation se construit à partir de l'association de plusieurs opérateurs. Chaque opérateur ayant deux entrées et une sortie, on utilise la sortie du premier opérateur comme entrée du suivant. Par exemple :

Équation A ET B ET C (A^B^C)



La zone en rouge montre un opérateur ET classique sans son mangeur de sortie. La sortie de cette porte, qui est ici le planeur de B, est associée à l'entrée C pour construire l'équation (A ET B) ET C, le mangeur de sortie est alors sur la trajectoire du planeur de C.

Ce mécanisme permet de générer l'ensemble des combinaisons possibles et donc les équations correspondantes. L'algorithme de LogiCell peut gérer un nombre quelconque d'entrées, l'applet n'est limitée à quatre entrées distinctes que pour des questions de simplicité et d'interface.

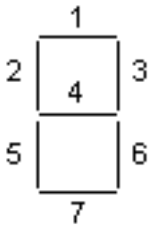
F- Construction des programmes

Les composants d'automatismes (algorithmes) gèrent des sorties multiples. Pour ce faire LogiCell associe plusieurs équations sur une grille unique. Le programme gère ensuite les différentes sorties en fonction du problème. On note ici le traitement parallèle des différentes équations.

LogiCell propose quelques applications combinatoires à titre d'illustration. Ce sont :

1. ***Le Va-Et-Vient*** : c'est le système classique d'inversion de l'état d'une ampoule gérée par deux interrupteurs : vous allumez votre couloir en sortant du salon, le traversez puis l'éteignez quand vous arrivez à la porte. En rentrant, vous faites exactement l'inverse. Chacun des interrupteurs sert une fois à l'allumage, une fois à l'extinction. On a ici deux entrées (les deux interrupteurs) et une sortie, soit une seule équation. En l'occurrence : Non (A Xor B) ; ~(AwB).
2. ***L'additionneur binaire 1 bit*** : on dispose de deux bits d'entrée (A et B) et on les additionne. Le résultat s'exprime sur deux bits : un bit de somme et un bit de retenue. Pour ceux qui souhaitent comprendre les équations, je vous invite à visiter le superbe site de Patrick Trau, à http://www-ipst.u-strasbg.fr/pat/autom/.
   Les deux équations sont :
   1. Somme : AwB.
   2. Retenue : A^B.
3. ***L'afficheur*** : Les anciens écrans à cristaux liquides affichaient les caractères à partir d'une matrice de sept segments.

Afficheur sept segments.	Représentation du chiffre 5.



La valeur à afficher est construite en binaire à partir des quatre bits d'entrée sous la forme DCBA, A étant le bit de poids faible. Par exemple la configuration D faux, C vrai, B faux, A vrai vaut 0101 binaire soit 5 en décimal. Pour afficher le chiffre 5, les segments 12467 doivent être allumés. Chacun des segments est géré par une équation :

1. Segment 1: ~(AwB)vCvD.
2. Segment 2: (C^~D^~(A^B))vDv(~A^~B^~C^~D).
3. Segment 3: ~(AwB)v~C.
4. Segment 4: (B^C^~A)v(BwC)vD.
5. Segment 5: (~CvB)^~A.
6. Segment 6: AvCv~BvD.
7. Segment 7: (~AvBvCvD)^(AvBv~CvD)^(~Av~Bv~CvD).

Les équations de 2, 4 et 7 sont particulièrement longues. Pour accélérer le traitement, dans le premier mode Afficheur, ces segments sont calculés par le programme, l'automate ne gère alors que les quatre segments restant. Le mode Afficheur 2 traite l'ensemble des sept segments.

4. ***L'additionneur binaire 2 bits*** : Le principe est le même que pour l'additionneur précédent mais on gère des nombres de 2 bits. On fait alors la somme BA+DC, le résultat est sur trois bits. Par exemple, B vrai et A faux donnent 10 binaire soit 2 en décimal, D faux et C vrai donnent 01 binaire soit 1 en décimal. La somme 10+01 binaire (2+1 décimal) vaut 101 binaire (3 en décimal). Les trois bits de sorties sont :
    1. La somme des deux bits de poids faible A+C, soit :
       AwC.
    2. La somme des deux bits de poids fort plus l'éventuelle retenue de A+C, soit :
       DwBw(A^C).
    3. *La* retenue des sommes précédentes soit :
       (B^D)v((BwD)^(A^C)).

LogiCell est indéniablement un peu technique, son rapport à la vie artificielle n'est pas direct. Toutefois il montre une partie essentielle de ce que l'on entend lorsque l'on dit qu'un automate de Conway, un jeu de la vie, est une machine de Turing.

1- Les personnes intéressées par le sujet peuvent se reporter à : Rivenc F., *Introduction à la logique*, Payot, Paris, 1989.

2- Heudin JC, *L'évolution au bord du chaos*, Hermès, Paris, 1998, p. 122. On note que les opérateurs NAND (non et) et NOR (non ou) permettent également chacun de construire l'ensemble des fonctions booléennes.

3- Patrick Trau, IPST, Université Louis Pasteur Strasbourg.
http://www-ipst.u-strasbg.fr/pat/autom/.

Jean-Philippe Rennard 11/2000.
http://www.rennard.org/alife
alife@rennard.org