Tableaux de Karnaugh en LaTeX (avec l'aide de Scilab ...)

Ph. ROUX

29 mars 2009

Table des matières

1 LaTeX et les tableaux de Karnaugh

1.1 le package `kvmacros.tex`

Dessiner le tableau de karnaugh d'une fonction booléenne à 4 ou 5 variables à l'aide d'un traitement de texte n'est pas chose aisée. Cependant, pour les utilisateurs de LaTeX, il existe un package qui permet de simplifier l'écriture de tels tableau, le package `kvmacros` de A.W. Wieland.

En fait il s'agit d'un simple fichier `kvmacros.tex` contenant un ensemble de commandes pour produire des tableaux de Karnaugh respectant les conventions les plus usitées. Il suffit de mettre dans le préambule du document `\input kvmacros` pour pouvoir utiliser ces commandes. Ensuite pour insérer un tableau de Karnaugh il faudra appeler la commande `karnaughmap` avec la syntaxe suivante :

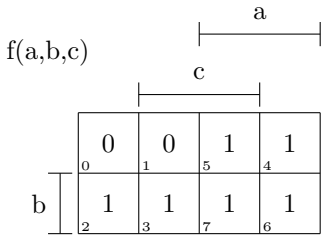
\karnaughmap{*nbre_var*}{*nom_fonc*}{*var*}{*val*}{*commandes graphiques*}

- *nbre_var* = nombre de variables de la fonction booléenne,
- *nom_fonc* = nom que l'on donne à la fonction dont on va tracer le tableau de Karnaugh (f par exemple),
- *var* = nom de chaque variable (les unes à la suite des autres sans espaces)
- *val*=ensemble des valeurs de f dans l'ordre des monômes canoniques,
- *commandes graphiques*= ensemble de commandes pour dessiner les groupements.

Quelques remarques à propos des différents paramètres :

- *nbre_var* est un entier,
- *nom_fonc* est une chaîne de caractères ou éventuellement une équation entre $...$,
- si les noms des variables sont a,b,c on écrira dans le champ *var* la chaîne `abc` (ou plutôt {`$a$`}{`$b$`}{`$c$`} si on veut écrire en mode mathématique), par contre si on veut des variables plus élaborées comme x_1,x_2,x_3 on écrira {`${x_1}$`}{`${x_2}$`}{`${x_3}$`}
- Pour l'argument *val* si on fait apparaître les 0 on notera la suite 100111100 ... pour cacher les 0 on les remplacera par des $\sim$: `1~~1111~~`...,
- Pour insérer des objets graphiques sur le tableau (cercles, ovales , ...), dans le but de dessiner les groupements, on peut utiliser les commandes de LaTeX en ajoutant éventuellement des packages comme `color`, `epic` et `eepic`...
- Pour finir on ne peut noter qu'on a pas à spécifier la position des flèches ... normalement c'est LaTeX qui s'occupe de tout!

Prenons un exemple, pour obtenir le tableau de Karnaugh suivant :



on a utilisé la commande :

```
\karnaughmap{3}{f(a,b,c)}{abc}{00111111}{}
```

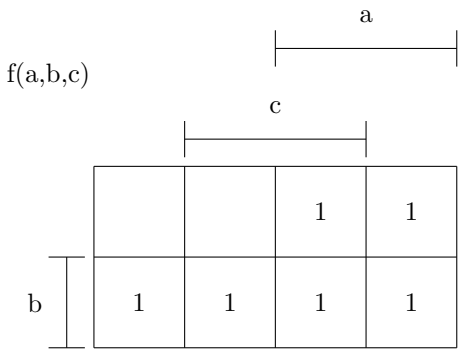
On peut personnaliser l'affichage du tableau à partir des commandes :

- `\kvnoindex` qui bloque l'affichage des numéros des cases du tableau (correspondants au monôme canonique associé à la case).
- `\kvunitlength` (qui est en fait une variable) permet de fixer la taille d'une case (en mm, pts, pouces ...) et donc d'ajuster la taille d'affichage du tableau.

En reprenant l'exemple précédent :

```
\kvnoindex
\kvunitlength=12mm
\karnaughmap{3}{f(a,b,c)}{abc}{~~111111}{}
```

on obtient :



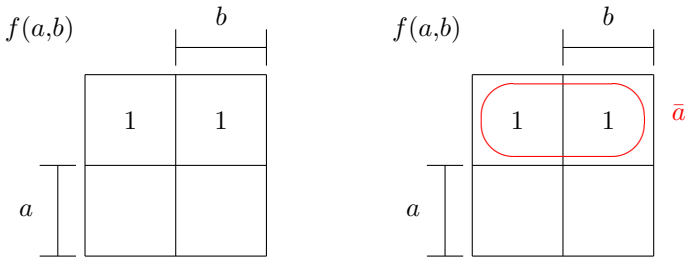
1.2 tracer les groupements

Il ne reste donc plus qu'à placer les groupements sur le tableau de Karnaugh en utilisant les capacités graphiques de LaTeX. Voici quelques exemples de commandes très simples à lire :

```
\put(x_pos,y_pos){object(x_dim,y_dim)}
```

- `x_pos` et `y_pos` désignent les coordonnées du centre de l'`objet`,
- l'`objet` que j'utilise est en général un `oval` mais on peut en faire d'autres,
- `x_dim` et `y_dim` désignent les dimensions de l'`objet` (on peut aussi mettre juste du texte),
- le coin en bas à gauche du tableau a pour coordonnées (0,0) et chaque case est un carré de coté unité (longueur 1).

Par exemple pour entourer le groupement $\bar{a}$ dans ce tableau

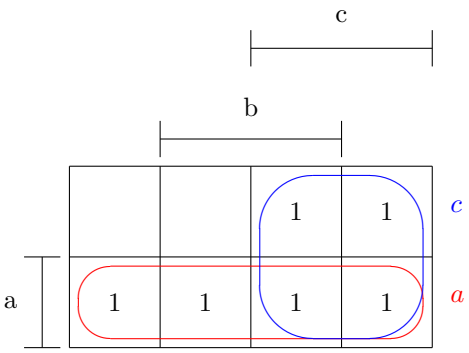


il a suffit de placer un `oval` rouge centré sur les coordonnées `x_pos`=1 et `y_pos`=1.5, de largeur un peu inférieure à 2 (1.8 ici) et de longueur un peu inférieure à 1 (0.8 ici), et de placer le texte $\bar{a}$ à coté (en position `(2.2,1.5)`) :

```
\kvnoindex
\kvunitlength=12mm
\karnaughmap{2}{$f(a,b)$}{{$a$}{$b$}}{11~~}
{
\put(1,1.5){\color{red}\oval(1.8,0.8)}
\put(2.2,1.5){\color{red}$\bar a$}
}
```

Si on trouve que l'ordre des variables n'est pas « naturel »on peut le modifier mais cela influe sur la position des groupements :

```
\karnaughmap{3}{}{cab}{~~111111}
{
\put(2,0.5){\color{red}\oval(3.8,0.8)}
\put(4.2,0.5){\color{red}$a$}
\put(3,1){\color{blue}\oval(1.8,1.8)}
\put(4.2,1.5){\color{blue}$c$}
}
```



2 Utilisation de Scilab

Ces packages permettent donc de créer facilement des tableaux de Karnaugh ... sauf qu'il est vite assez pénible de faire la liste des valeurs de la fonction dans l'ordre des monômes canoniques ou de rentrer manuellement toutes les informations sur les groupements (sans compter les erreurs de saisie). C'est pour cette raison que j'ai écrit quelques fonctions Scilab pour automatiser ce travail! Je vais décrire ci-dessous les principales fonctions contenues dans le fichier `Karnaugh.sce` qui permettent de résoudre ces problèmes. Le code est loin d'être optimal, mais il marche et il est suffisamment commenté pour donner des idées au lecteur intéressé.

2.1 Transcriptions des fonctions booléennes

Il faut d'abord fixer quelques conventions d'écriture pour représenter les expressions booléennes. L'idée est de représenter une fonction :

$$f(a,b,c,d) = \overline{a} + bc + (a+b)\overline{(c+d)} + c \times a$$

par la chaîne de caractère :

```
_a+bc+(a+b)_(c+d)+c*a
```

on considèrera donc que :

- chaque variable est composée d'une et une seule lettre (`a,b,c`... je doute que quelqu'un ait réellement besoin de faire des tableaux de Karnaugh à plus de 26 variables!)
- + représente l'addition, * la multiplication et _ placé devant une expression représente le complémentaire,
- on peut utiliser les parenthèses `( )` et omettre la multiplication * dans les monômes `abc` ou entre des parenthèses `(a+b)(c+d)`.

Scilab permet de faire du calcul booléen avec la syntaxe suivante :

- les deux booléens sont vrai : `%T` (le 1) et faux : `%F` (le 0)

– les trois opérateurs de base sont : `&` qui correspond à la conjonction ($\times$), `|` qui correspond à la disjonction ($+$) et $\sim$ qui correspond à la négation (la barre $\bar{}$),

Maintenant pour pouvoir manipuler les fonctions booléennes avec scilab, il faut construire une fonction qui permette de traduire la chaîne de caractères :

`_a+bc+(a+b)_(c+d)+c*a`

en une expression booléenne scilab :

`(~a)|(b&c)|(((a)|(b))&~((c)|(d)))|(c&a)`

C'est ce qui est réalisé par la fonction `boolean_2_scilab` qui prend en entrée une chaîne de caractère représentant une expression booléenne et renvoie en sortie une autre chaîne de caractères représentant la traduction de cette expression booléenne en langage scilab. Cette fonction ne se contente pas de remplacer les `+` par `|` ou `*` par `&`, elle doit surtout gérer les problèmes de parenthésage liés aux conventions de priorités entre les opérateurs `+` et `*` énoncées plus haut.

2.2 Évaluation des fonctions booléennes

Une fois la fonction booléenne traduite en expression scilab, il faut pouvoir évaluer cette fonction afin de remplir son tableau de Karnaugh. Si on a la liste des variables `a,b,c` ... et l'expression booléenne de la fonction sous forme de chaînes de caractères `var` et `expr_scilab` il est facile de créer la fonction scilab associée via la commande :

```
deff('bool=f('+var')','bool='+expr_scilab)
```

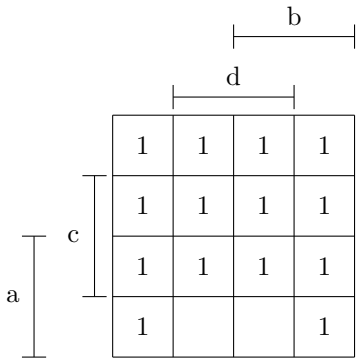
il faut ensuite évaluer `f` pour tous les choix de valeurs possibles pour les variables `a,b,c` ... Pour n variables il y a 2^n possibilités, qui correspondent à l'écriture binaire des nombres de 0 à 2^{n-1}. J'ai donc écrit une fonctions `[E]=binaire(e,w)` qui écrit dans un tableau `E` l'entier `e` sous forme binaire avec `w` bits, mais avec `%t` à la place de 1 et `%f` à la place de 0. Ensuite la fonction `calcule_tab_karnaugh` crée la fonction booléenne et appelle autant de fois que nécessaire la fonction `binaire` pour remplir le tableau de Karnaugh :

```
-->vars=['a' 'b' 'c' 'd']//ordre par défaut des variables
 vars  =
!a  b  c  d  !
-->expr_scilab='~(a&d&(~c))'
 expr_scilab  =
 ~(a&d&(~c))
-->calcule_tab_karnaugh(vars,expr_scilab)
 ans  =
 111111111~111~11
 -->vars=['a' 'c' 'b' 'd']//on change l'ordre des variables
 vars  =
!a  c  b  d  !
-->calcule_tab_karnaugh(vars,expr_scilab)
 ans  =
```

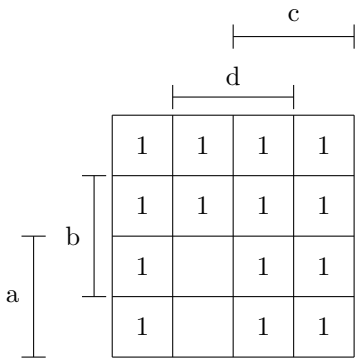
```
 111111111~1~1111
```

les résultats précédents permettent de construire les tableaux de Karnaugh :

```
\karnaughmap{4}{}{abcd}{111111111~111~11}{}
```




```
\karnaughmap{4}{}{acbd}{111111111~1~1111}{}
```



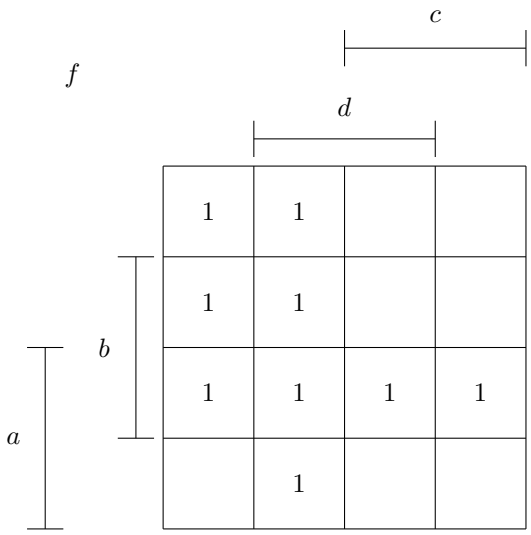
On peut enfin écrire une fonction qui génére l'ensemble du code LaTeX relatif au tableau de Karnaugh :

```
-->[latex]=tab_karnaugh('f','ab+_c(d+_a)',vars)
 latex  =

!$f(a,b,c,d)=ab+\bar c{(d+\bar a)}$ \\       !
!                                            !
!\kvnoindex                                   !
!                                            !
!\kvunitlength=12mm                           !
!                                            !
!%généré par la commande scilab :             !
!                                            !
!%tab_karnaugh('f','ab+_c(d+_a)',vars)        !
!                                            !
!\karnaughmap{4}{$f$}{{$a$}{$c$}{$b$}{$d$}}  !
!                                            !
!{1111~~~~~111~~11}                           !
!                                            !
!{%pas de groupements dessinés                !
!                                            !
!}                                            !
```

qui produira le tableau suivant :

$f(a,b,c,d) = ab + \bar{c}(d + \bar{a})$



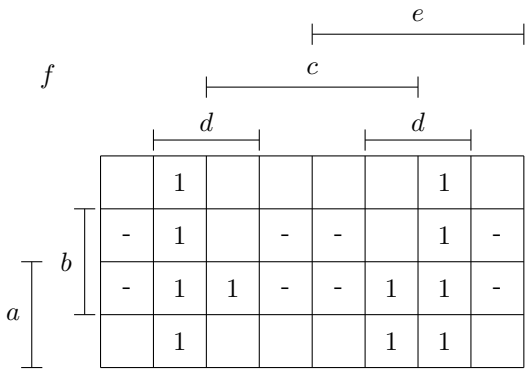
Quelques efforts supplémentaires permettent de tracer des fonction « phi-booléennes » [1]. J'ai alors besoin de deux expressions booléennes : une pour définir la fonction (sur son domaine) une autre pour définir les cases où la fonction n'est pas définie. Il suffit ensuite d'utiliser `calcule_tab_karnaugh` pour savoir où positionner les 1 dans le tableau de Karnaugh et où placer les caractères signalant que la fonction n'est pas définie (`-` pour moi, parfois aussi ϕ est utilisé). Voici un exemple avec une fonction à 5 variables :

```
-->latex=tab_karnaugh_phi('f','ab+_cd+be_d+acde','_db',vars)
 latex  =
!                                                                    !
!                                                                    !
!\kvnoindex                                                          !
!                                                                    !
!\kvunitlength=12mm                                                  !
!                                                                    !
!%généré par la commande scilab :                                    !
!                                                                    !
!%tab_karnaugh_phi('f','ab+_cd+be_d+acde','_db',[ 'e'  'a'  'c'  'b'  'd'!
!      ])                                                            !
!                                                                    !
!\karnaughmap{5}{$f$}{{$e$}{$a$}{$c$}{$b$}{$d$}}                     !
!                                                                    !
!{~1-1~~-~~1-1~~-1~1-1~~-~~1-1~1-1}                                  !
```

1. les informaticiens ont la très mauvaise habitude d'appeler "fonction booléennes" les applications à valeur booléennes, il sont donc obligés d'utiliser une autre dénomination pour les vraies fonctions booléennes(*i.e.* avec un ensemble de définition plus petit que l'ensemble de départ) qui appellent donc fonctions *phi-booléennes* (ou ϕ-booléennes).

```
!                                                                    !
!{%pas de groupements dessinés                                       !
!                                                                    !
!}                                                                   !
```

ce qui donne le tableau :



2.3 Dessiner les groupements

Le but de cette partie est de générer automatiquement les commandes graphiques pour entourer des groupements dans le tableau de Karnaugh. Comme c'est assez difficile je me suis restreint au cas des fonctions à n variables pour $n = 2,3,4,5$. la fonction `init_variables` permet d'initialiser, en fonction de n, un certains nombres de données nécessaires à la gestions des groupements :

```
-->[num_karnaugh, x_karnaugh, y_karnaugh, vars]=init_variables(3)
 vars  =

!b  a  c  !
 y_karnaugh  =

    2.    2.    2.    2.
    1.    1.    1.    1.
 x_karnaugh  =

    1.    2.    3.    4.
    1.    2.    3.    4.
 num_karnaugh  =

    0.    1.    5.    4.
    2.    3.    7.    6.
```

– `num_karnaugh` remplit une matrice scilab avec les numéros de case (par défaut) du tableau de Karnaugh,

- `x_karnaugh` et `y_karnaugh` renvoient des matrices scilab avec les numéros de colonne et de ligne associés à chaque case du tableau de Karnaugh (la numérotation démarre en bas à gauche),
- `vars` contient l'ordre « naturel »des variables

Si on veut étendre les possibilités des scripts suivants à plus de 5 variables il suffira de modifier `init_variables` pour qu'elle initialise correctement les variables précédentes.

Le problème a été découpé en plusieurs sous-problèmes, associés aux fonctions suivantes :

- `b=convert_base_2(a,n)` et `x=convert_base_10(a)` permettent de passer d'une représentation décimale d'un entier (en fait un réel pour scilab) à une représentation binaire (tableau de 0 et de 1) et inversement,

```
-->b=convert_base_2(21,5)
 b  =

    1.    0.    1.    0.    1.

-->x=convert_base_10(b)
 x  =
    21.
```

- `L=case_adjacente(une_case,n)` permet de connaître les cases adjacentes à une autre (suivant le nombre de variables n),

```
-->L=case_adjacente(0,4)
 L  =
    1.    2.    4.    8.
```

- `txt=nom_groupe(G,vars)` calcule le nom (en LaTeX) du monôme qui correspond au groupement

```
-->txt=nom_groupe([0 1 5 4 8 9 13 12],vars)
 txt  =
 \bar c
```

- `bool= cases_se_touchent(case1,case2,x_karnaugh,y_karnaugh)` permet de savoir si deux cases se touchent dans un tableau de Karnaugh (important pour dessiner les groupements en plusieurs morceaux),
- `L=parties_connexe(G,x_karnaugh,y_karnaugh)` permet d'isoler les parties connexes d'un groupement, exemple d'un groupement $\bar{c}$ en 2 parties :

```
-->L=parties_connexe([0 1 5 4 8 9 13 12],x_karnaugh,y_karnaugh)
 L  =
       L(1)

    5.    4.    1.    0.

       L(2)
```

```
        13.    12.    9.    8.
```

Au final on a une fonction `entoure_groupement` qui permet d'entourer un groupement donné par la liste des numéros de cases qui le composent. Par exemple avec le groupement $\bar{c}$ (en 2 morceaux) :

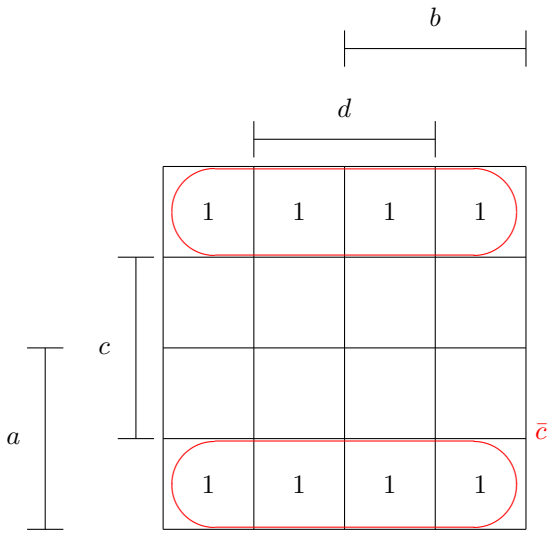
```
-->txt=nom_groupe([0 1 5 4 8 9 13 12],['a' 'b' 'c' 'd'])
 txt  =

 \bar c

-->latex=entoure_groupement([0 1 5 4 8 9 13 12],x_karnaugh,y_karnaugh,'red','\bar c')
 latex  =

!\color{red}\put(4.1,1){$\bar c$}  !
!                                  !
!\put(2,3.5){\oval(3.8,0.95)}      !
!                                  !
!\put(2,0.5){\oval(3.8,0.95)}      !
```

qui donne bien :



2.4 Recherche des monômes maximaux

Cette partie est la plus complexe, le but est de récupérer la liste des groupements maximaux automatiquement pour pouvoir ensuite tracer ces divers groupements sur le tableau de Karnaugh. Reconnaître un groupement dans le tableau de Karnaugh d'une fonction booléenne est facile si on connaît la relation

d'ordre usuelle sur l'algèbre de bool des fonctions booléennes définie par :

$$\forall f,m : \mathbb{B}^n \longrightarrow \mathbb{B},\ \ m \prec f \Longleftrightarrow m = f \times m$$

en effet si une fonction booléenne peut s'écrire comme somme de monômes alors :

$$f(a,b,c,\dots) = \sum_{i \in I} m_i(a,b,c,\dots) \Longrightarrow \forall i \in I, m_i \times f = m_i$$

On peut donc vérifier qu'un groupe appartient au tableau de Karnaugh en utilisant ce test, que l'on peut mettre en œuvre en faisant le produit (terme à terme) des tables de valeurs de m et f. Ensuite pour construire un monôme maximal à partir d'un monôme donné (canonique par exemple) on va appliquer récursivement la règle de simplification suivante :

si pour une variable $x \in \{a,b,c,\dots\}$ on a $x \times m \prec f$ et $\bar{x} \times m \prec f$ alors $m \prec f$

donc en partant d'un monôme (canonique) qui apparaît dans le tableau de Karnaugh, on peut construire des monômes de plus en plus « gros »en regardant pour chaque variable si en la modifiant en son complémentaire on obtient un nouveau groupement du tableau. Pour représenter un monôme j'utilise un tableau à n cases `bin` tel que :

- `bin(i)`=1 si la ième variable apparaît dans le monôme
- `bin(i)`=0 si la ième variable n'apparaît pas dans le monôme
- `bin(i)`=-1 si la négation de la ième variable apparaît dans le monôme

Par exemple le monôme $a\bar{b}d$ (à 4 variables) sera représenté par `[1 -1 0 1]` si l'ordre des variables est `d c b a`. En conséquence j'ai découpé le problème en sous-problèmes associés aux fonctions suivantes :

- `tab=table_valeurs(expr_scilab,vars)` calcule la table des valeurs (0 ou 1) d'une fonction booléennes définie par `expr_scilab` avec l'ordre des variables donné par `vars`,
- `expr_boolean=monome(bin,vars)` calcule l'expression booléenne (`ab_c` par exemple) associée à un monôme canonique `bin` est le numéro (tableau binaire) de la case correspondante du tableau de Karnaugh et `vars` donne l'ordre des variables,

```
-->vars
 vars  =
!d  b  c  a  !
-->expr_boolean=monome([1 0 1 0], vars)
 expr_boolean  =
 cd
```

- `bin=monome_max(une_case,expr_scilab,vars)` recherche du groupement maximal auquel appartient la case numéroté `une_case`, exemple :

```
-->bin=monome_max(15,'a',vars)
```

```
 bin  =
    0.    0.    0.    1.
-->monome(bin, vars)
 ans  =
 a
```

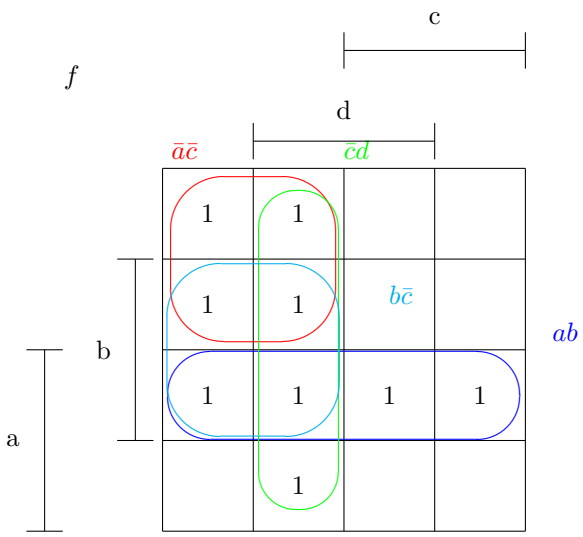
– la fonction `G=groupes_max(expr_scilab,vars,num_karnaugh)` permet de calculer une liste de groupements maximaux :

```
-->G=groupes_max(vars,'a|(~b)',num_karnaugh)
 G  =
!_b  a  !
```

Attention certains groupement maximaux peuvent être absent de cette liste prendre le cas de la fonction $f(a,b,c,d) = ab + \bar{c}d + \bar{a}\bar{c}$ on trouve : $f(a,b,c,d) = \bar{a}\bar{c} + ab + \bar{c}d$
):

```
-->G=groupes_max(vars,expr_scilab,num_karnaugh)
 G  =
!_a_c  ab  _cd  !
```

alors que le monôme $b\bar{c}$ est aussi maximal (mais pas obligatoire) :

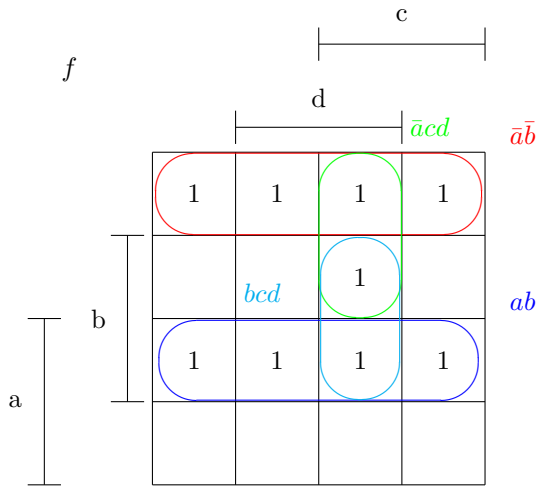


– Enfin la fonction `R=forme_reduite(vars,G,expr_scilab,num_karnaugh)` permet d'extraire une forme réduite de la fonction (définie par `expr_scilab`) à partir d'une liste de monômes maximaux (`G`) par exemple dans $f(a,b,c,d) = ab + \bar{a}\bar{b} + \bar{a}cd + bcd$ tous les monômes sont maximaux mais $\bar{a}cd$ et bcd ne sont pas obligatoire. Un seul d'entre eux est nécessaire à l'écriture d'une forme réduite de f :

```
-->forme_reduite(vars,['ab' '_a_b' '_acd' 'bcd'],expr_scilab,num_karnaugh)
 ans  =
```

```
!ab  _a_b  _acd  !
```

ce qu'on vérifie facilement sur le tableau de Karnaugh :



il ne reste plus qu'à intégrer la recherche des monômes maximaux et leur tracé à la fonction qui calcule le tableau de Karnaugh pour obtenir une implémentation complète de la méthode Karnaugh. C'est ce que fait la fonction `methode_karnaugh` :

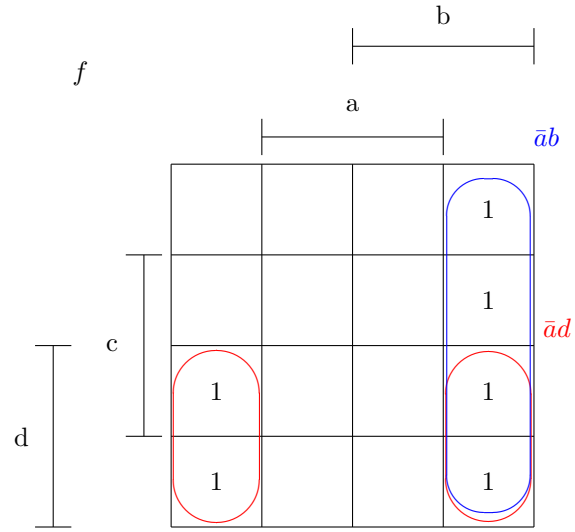
```
-->latex=methode_karnaugh('f','_a(b+cd)+_a_cd',4)
 latex  =

!$f(a,b,c,d)=\bar a{(b+cd)}+\bar a\bar cd$ \\                    !
!                                                                !
!\kvnoindex                                                      !
!                                                                !
!\kvunitlength=12mm                                              !
!                                                                !
!%généré par la commande scilab :                                !
!                                                                !
!%methode_karnaugh('f','_a(b+cd)+_a_cd',[ 'd'  'b'  'c'  'a' ])  !
!                                                                !
!\karnaughmap{4}{$f$}{dbca}                                      !
!                                                                !
!{~~~~1~1~1~1~1~1~}                                              !
!                                                                !
!{%les groupements                                               !
!                                                                !
!\color{red}\put(4.1,2.1){$\bar ad$}                             !
!                                                                !
!\put(0.5,1){\oval(0.95,1.9)}                                    !
!                                                                !
```

```
!\put(3.5,1){\oval(0.94,1.88)}                                    !
!                                                                 !
!\color{blue}\put(4,4.2){$\bar ab$}                               !
!                                                                 !
!\put(3.5,2){\oval(0.92,3.68)}                                    !
!                                                                 !
!}%fin des groupements                                            !
!                                                                 !
!\\                                                               !
!                                                                 !
!la forme simplifiée  est donc :\\                                !
!                                                                 !
!$f(a,b,c,d)=\bar ad+\bar ab$ \\                                   !
```

ce qui donne une fois compilé :

$f(a,b,c,d) = \bar{a}(b + cd) + \bar{a}\bar{c}d$



la forme simplifiée est donc :
$f(a,b,c,d) = \bar{a}d + \bar{a}b$

L'argument `vars` permet de spécifier l'ordre des variables, si on veut un ordre différent de l'ordre par défaut (celui renvoyé par `init_variables`), sinon on donne juste le nombre de variables.

3 Pour aller plus loin ...

Les fonction sont disponibles dans le Fichier `Karnaugh.sce` où ont été rajoutés quelques commandes ajoutant un menu à scilab pour tester de manière in-

teractive les fonctions `tab_karnaugh`, `tab_karnaugh_phi` et `methode_karnaugh`. Le code est loin d'être optimal et peut être facilement amélioré par le lecteur intéressé. Il permet cependant de voir comment utiliser scilab pour produire des corrigés d'exercices types en LaTeX en laissant scilab s'occuper des calculs mathématiques. On peut ensuite écrire le résultat dans un fichier avec la commande :

```
mputl(latex,'essai.tex')
```

et l'insérer dans un fichier LaTeXcompilant :

```
\documentclass{article}
\usepackage{amsmath,color}
\input kvmacros
\begin{document}
\pagestyle{empty}
\input essai.tex
\end{document}
```