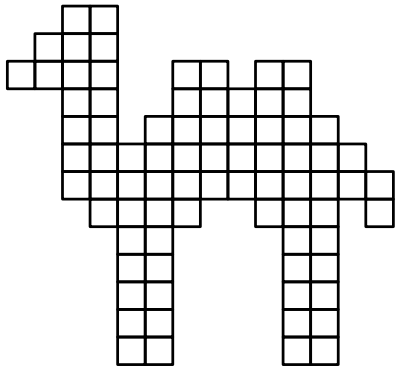
`combine` : une bibliothèque OCaml pour la combinatoire

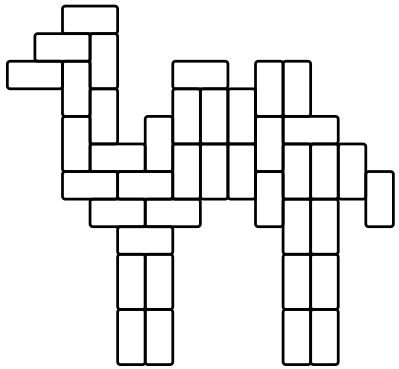
Rémy El Sibaïe Besognet
Jean-Christophe Filliâtre

JFLA 2013

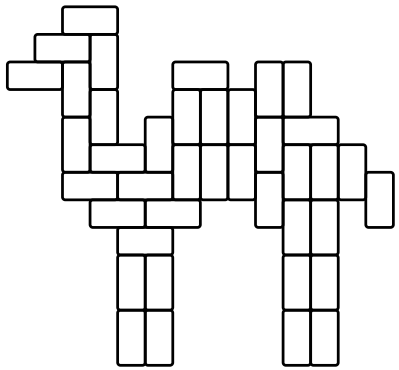
Pavage



Pavage

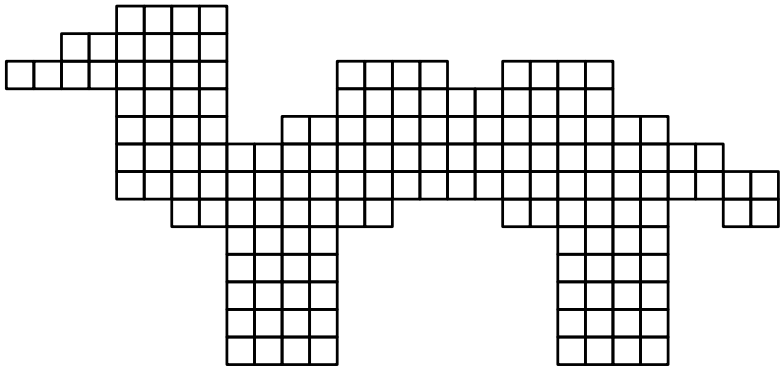


Pavage

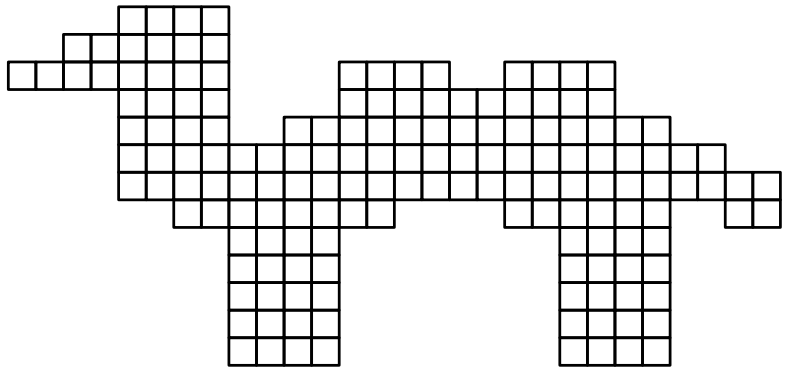


46 976 solutions

Pavage



Pavage



32 420 116 341 024 288 solutions

Couverture exacte de matrice (EMC)

soit une matrice contenant des 0 et des 1

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

on cherche un sous-ensemble de lignes
avec un 1 et un seul par colonne

Applications

de nombreux problèmes peuvent se ramener à EMC

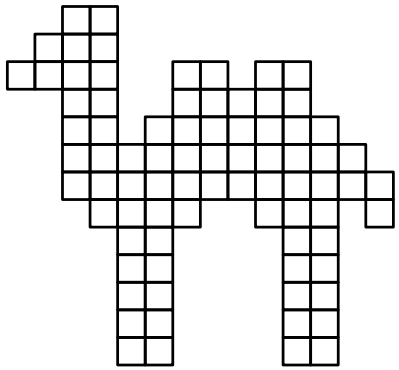
exemples :

- pavage
- N-reines
- Sudoku
- coloriage de graphe

Exemple : pavage avec des dominos 2×1

colonnes = cases à paver

lignes = différentes façons de poser un domino



$$
\begin{array}{cccccccccccccccc}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \ldots \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \ldots \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \ldots \\
 & & & & & & & & \vdots & & & & & & &
\end{array}
$$

Interface

```
type t

val create: ?primary:int → bool array array → t

type solution = int list

val iter_solution: (solution → unit) → t → unit
val get_first_solution: t → solution
val count_solutions: t → int
```

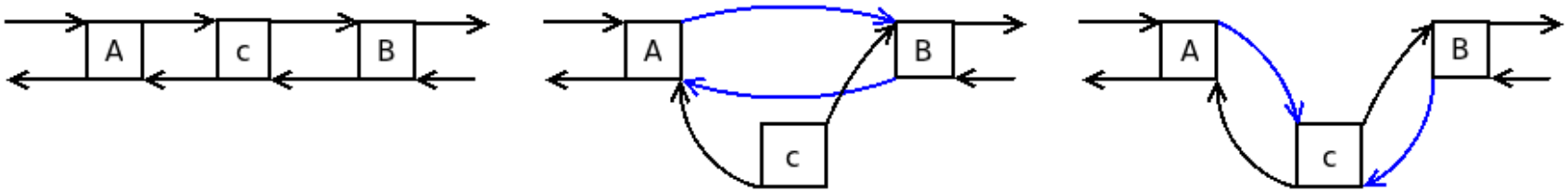
Deux techniques pour résoudre EMC

- les liens dansants (DLX)
- les *Zero-Suppressed BDDs* (ZDD)

Les liens dansants (DLX)

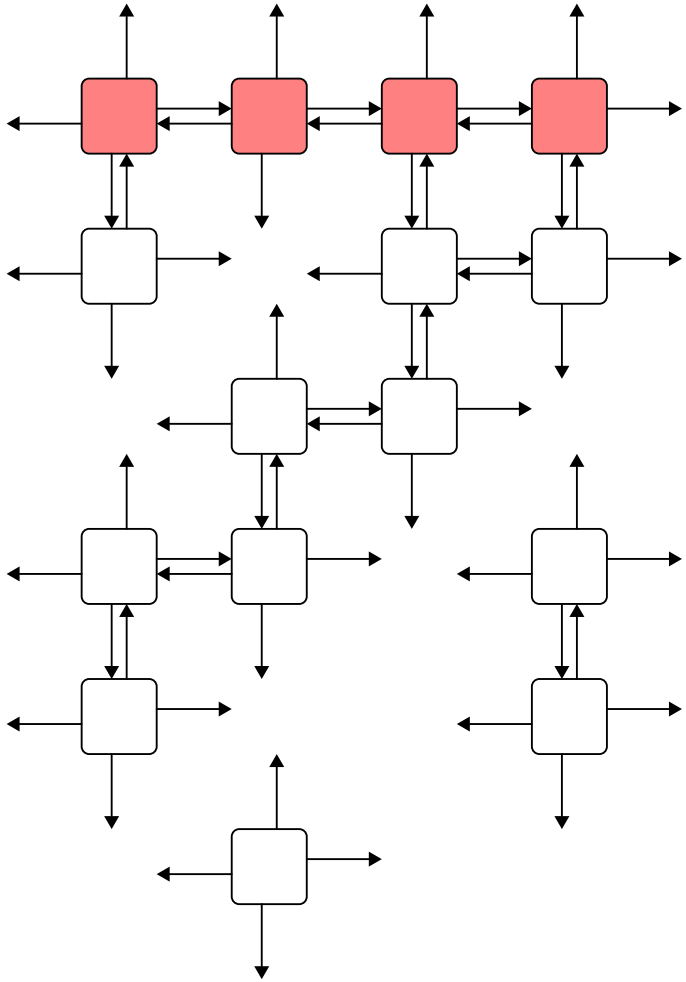
Donald Knuth, Dancing links
(Millenial Perspectives in Computer Science, 2000)

une utilisation astucieuse des listes doublements chaînées :



Les liens dansants (DLX)

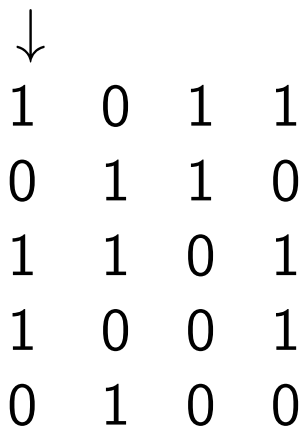
$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$



Les liens dansants (DLX)

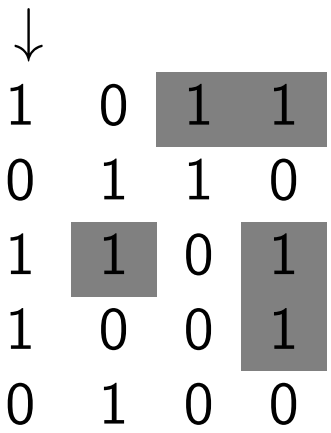
$$\begin{matrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{matrix}$$

Les liens dansants (DLX)



on choisit une colonne

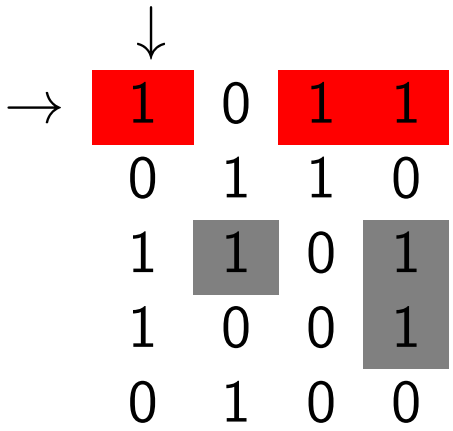
Les liens dansants (DLX)



on la couvre

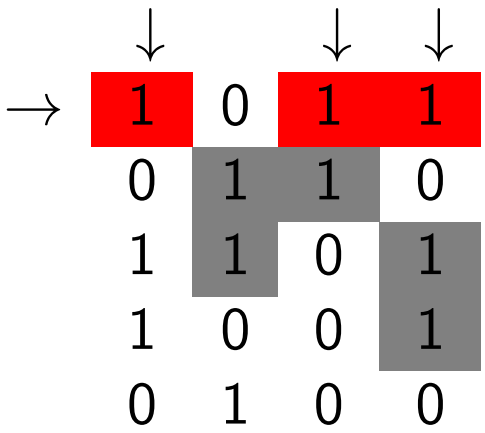
i.e. on efface les 1 sur les mêmes lignes que ses 1

Les liens dansants (DLX)



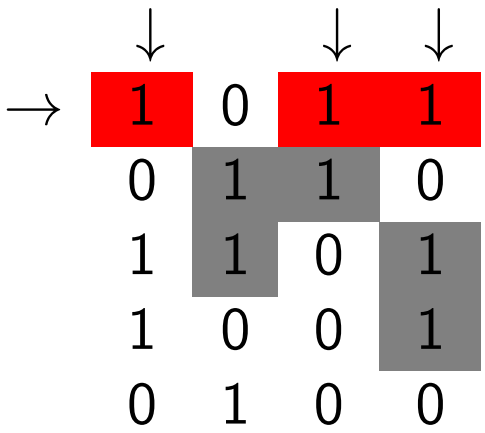
on parcourt successivement ses 1

Les liens dansants (DLX)



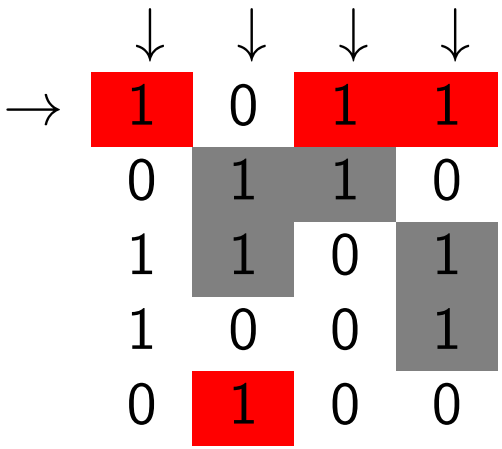
on couvre les colonnes de la ligne sélectionnée

Les liens dansants (DLX)



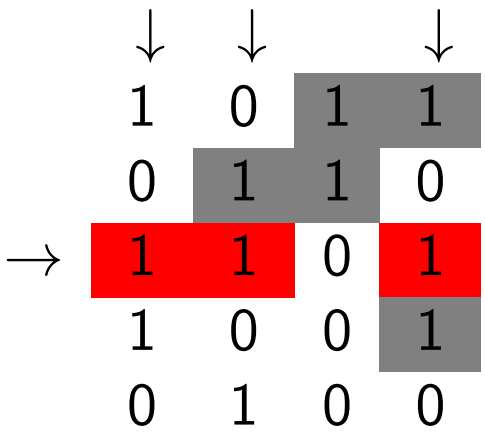
on recommence tant qu'il reste des colonnes à couvrir

Les liens dansants (DLX)



on obtient ici la solution $\{0,4\}$

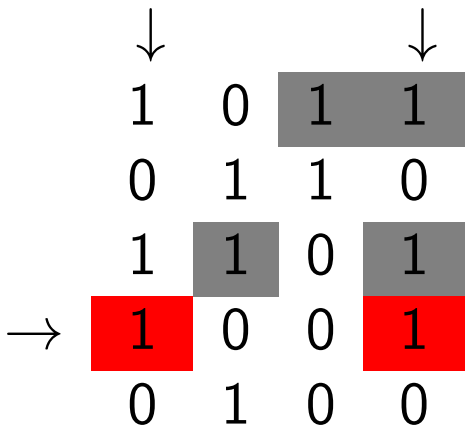
Les liens dansants (DLX)



on *backtrack* et on passe au 1 suivant de la première colonne

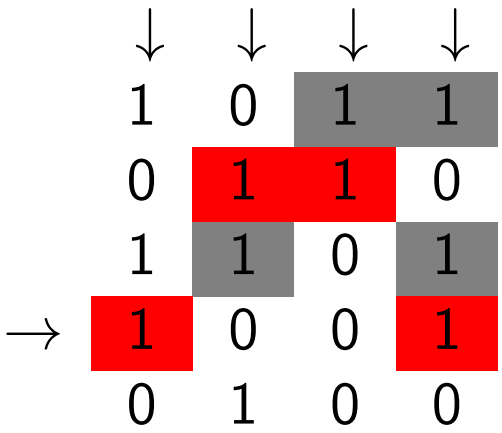
impossible de couvrir la troisième colonne $\Rightarrow$ pas de solution

Les liens dansants (DLX)



on *backtrack* encore et on passe au dernier 1 de la première colonne

Les liens dansants (DLX)



une seule façon de couvrir la troisième colonne

donne la solution {1,3}

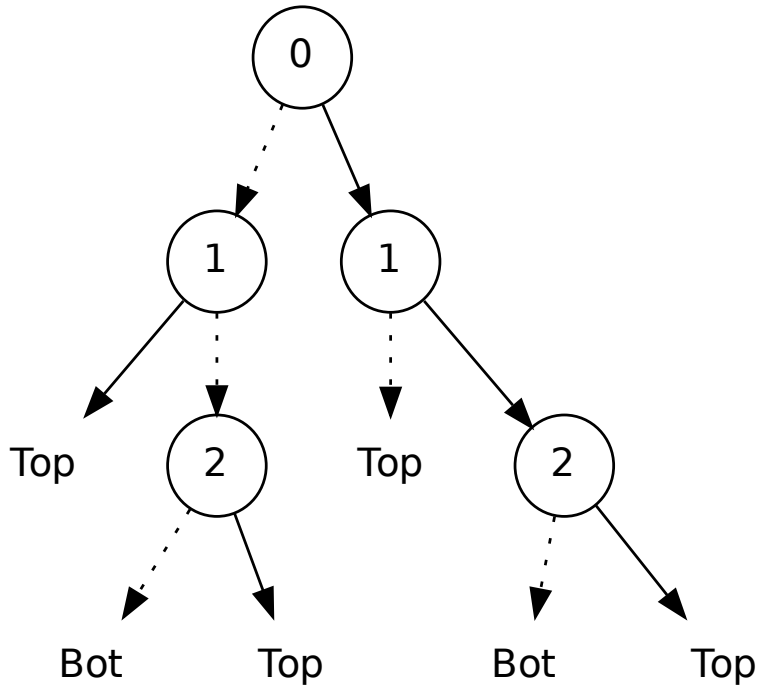
Et maintenant... ZDD

Zero-Suppressed Binary Decision Diagram (ZDD)

Shin ichi Minato (DAC, 1993)

ensemble d'ensembles d'entiers

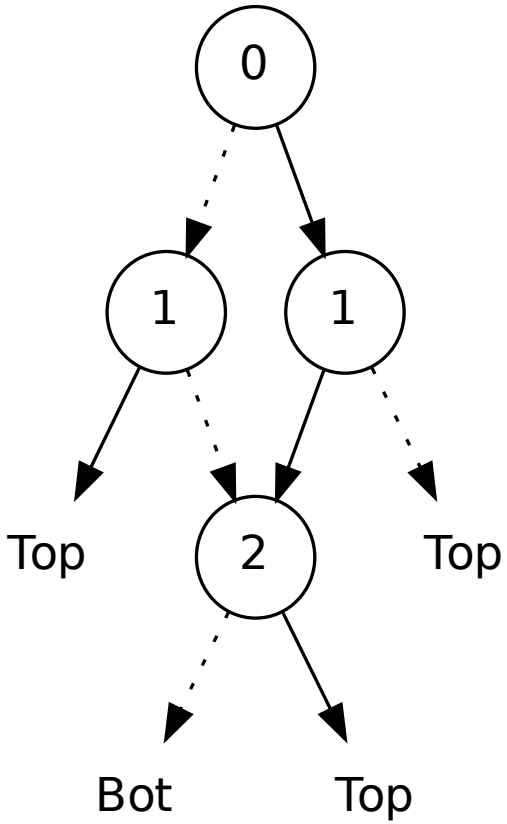
{{0, 1, 2}, {0}, {1}, {2}}



Comme pour les BDD

- partage maximal des sous-arbres
- ordre croissant des éléments
- pas de Bot à droite

- opérations avec mémoïzation

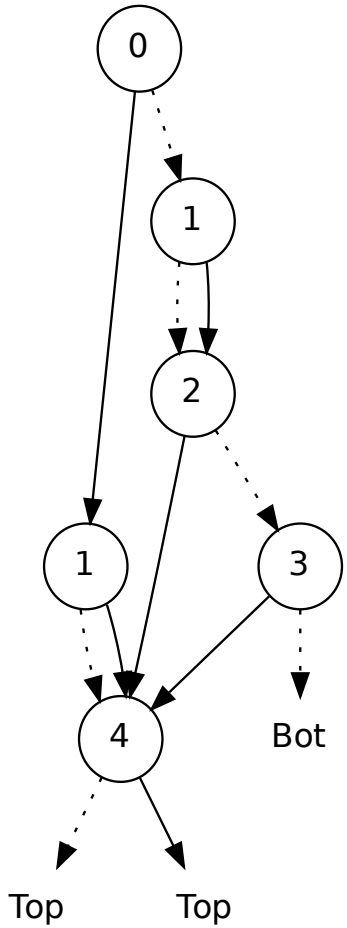


EMC vers ZDD

d'abord la couverture d'une colonne

$$
\begin{array}{ll}
0. & \left(\begin{array}{cccc} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{array}\right) \\
1. & \\
2. & \\
3. & \\
4. &
\end{array}
$$

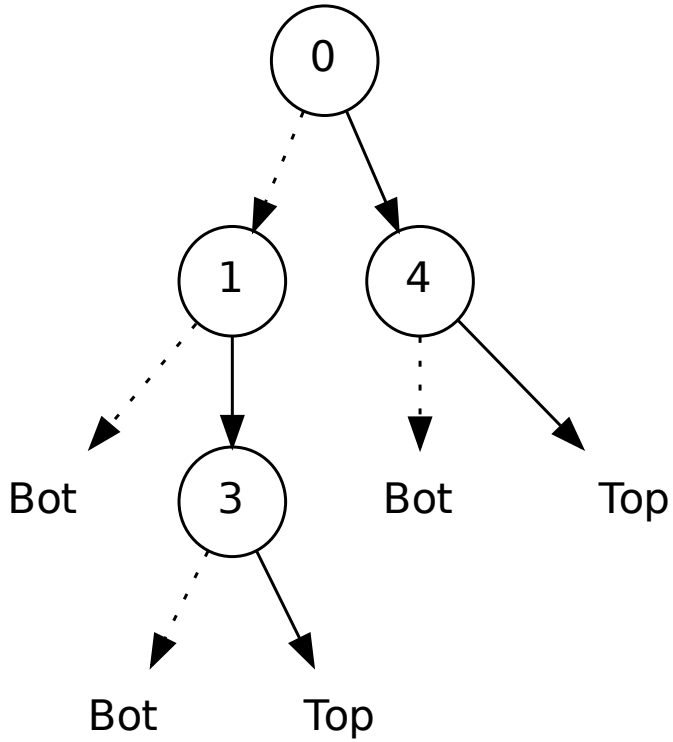
$\{\{0\},\{0,1\},\{0,4\},\{0,1,4\},$
$\{2\},\{2,1\}, \ldots\}$



EMC vers ZDD

on fait ensuite
l'intersection
de tous ces ZDD

$$
\begin{array}{ll}
0. & \left(\begin{array}{cccc} 1 & 0 & 1 & 1 \end{array}\right. \\
1. & \begin{array}{cccc} 0 & 1 & 1 & 0 \end{array} \\
2. & \begin{array}{cccc} 1 & 1 & 0 & 1 \end{array} \\
3. & \begin{array}{cccc} 1 & 0 & 0 & 1 \end{array} \\
4. & \left.\begin{array}{cccc} 0 & 1 & 0 & 0 \end{array}\right)
\end{array}
$$



Comparaison

DLX
VS
ZDD

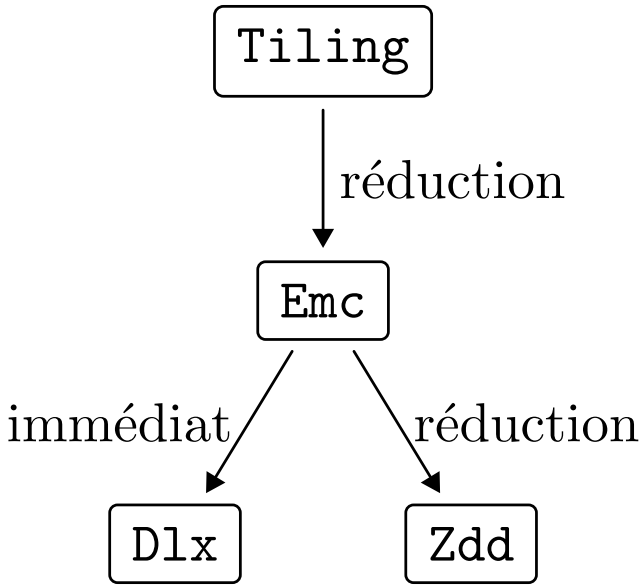
DLX

+ espace constant
+ efficace pour trouver une solution
– trouve les solutions une à une

ZDD

+ construit toutes les solutions
+ dénombrement facile
– nécessite beaucoup de mémoire

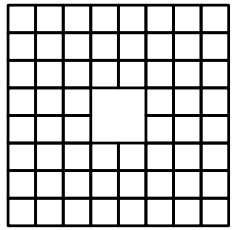
La bibliothèque `combine`



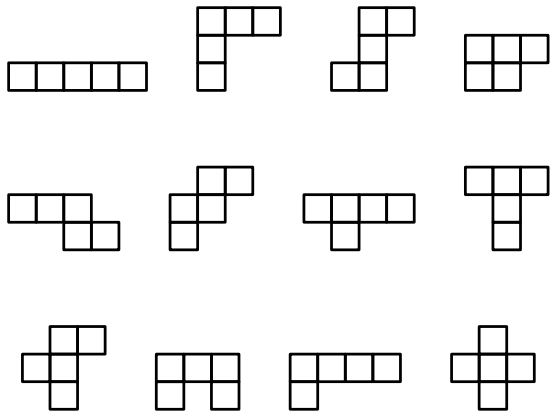
+ un mini langage pour le pavage

Exemple : le problème de Scott (1958)

combien y a-t-il de façons de paver ces 60 cases



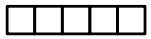
avec les douze pentaminos ?



Exemple : le problème de Scott

on commence par décrire les 12 pentaminos

```
pattern I = {*****}
```

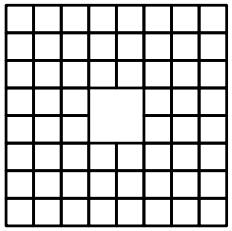


```
pattern V = {
***
*
*
}
...
```

```
tiles pentaminos =
  [ L ~one ~sym, T ~one ~sym, V ~one ~sym, N ~one ~sym,
    Z ~one ~sym, F ~one ~sym, X ~one ~sym, W ~one ~sym,
    P ~one ~sym, I ~one ~sym, Y ~one ~sym, U ~one ~sym ]
```

Exemple : le problème de Scott

puis on décrit la surface à paver



```
pattern scott_board =
  set set set set
    (constant 8x8 true)
    3x3 false 3x4 false 4x3 false 4x4 false
```

et enfin on dénombre les solutions

```
problem scott_problem = scott_board pentaminos
count dlx scott_problem
```

Exemple : le problème de Scott

on peut exploiter les symétries du problème

en posant la pièce  toujours de la même façon

```
tiles pentaminos =
  [ L ~one ~sym, T ~one ~sym, V ~one ~sym, N ~one ~sym,
    Z ~one ~sym, F ~one     , X ~one ~sym, W ~one ~sym,
    P ~one ~sym, I ~one ~sym, Y ~one ~sym, U ~one ~sym ]
```

on trouve alors 65 solutions (distinctes)

Conclusion

résumé

- deux modules DLX et ZDD
- une interface simple et commune (EMC)
- un langage pour le pavage en dimension 2

`http://www.lri.fr/~filliatr/combine/`

perspectives

- d'autres problèmes de pavage (3D, non rectangulaire, etc.)
- prendre en compte les symétries du problème