Vérification formelle de programmes de génération de données structurées
Résumé des travaux de thèse de doctorat

Richard Genestier
FEMTO-ST, UMR CNRS 6174 (UBFC/UFC/ENSMM/UTBM)
Université de Franche-Comté, France
richard.genestier@femto-st.fr

AFADL 2016

Sujet

Vérification formelle de programmes de génération de données structurées

- Directeur : Olga Kouchnarenko, professeur
- Encadrant : Alain Giorgetti, maître de conférences

Contexte

- Conception et implémentation en C d'algorithmes efficaces
- Structures de données composées de tableaux
- **Vérification déductive** de programmes paramétrés par des tableaux
- Automatisation privilégiée

Contexte

- Conception et implémentation en C d'algorithmes efficaces
- Structures de données composées de tableaux
- **Vérification déductive** de programmes paramétrés par des tableaux
- Automatisation privilégiée
- Outils
  - Frama-C : plateforme d'analyse de code C dévelopée par le CEA LIST et INRIA Saclay
  - Langage de spécification ACSL pour annoter des programmes C
  - Greffon WP pour "Weakest Precondition calculus"
  - Génération de conditions de vérification avec Why3
  - Appels de solveurs SMT (Alt-Ergo, CVC3, CVC4)
- Compléments : Preuves interactives, test exhaustif borné, test aléatoire

Motivations

- Faisabilité des preuves automatiques de programmes
- Identification des possibilités des outils de vérification automatique
- Illustration de l'intérêt de la spécification formelle

Motivations

- Faisabilité des preuves automatiques de programmes
- Identification des possibilités des outils de vérification automatique
- Illustration de l'intérêt de la spécification formelle
- Cadre d'investigation : combinatoire énumérative
  - Nouveau domaine d'application
  - Nombreux algorithmes adaptés aux prouveurs
  - **Cartes combinatoires**
- Différentes études de cas

Plan

Bibliothèque de générateurs séquentiels vérifiés

Permutation

- Combinatoire : partie des mathématiques discrètes qui étudie des ensembles finis ou dénombrables d'objets
- Combinatoire énumérative : algorithmes pour
  - générer ces objets
  - compter ces objets

Permutation

- Combinatoire : partie des mathématiques discrètes qui étudie des ensembles finis ou dénombrables d'objets
- Combinatoire énumérative : algorithmes pour
  - générer ces objets
  - compter ces objets
- Exemple de structure combinatoire : Permutations de n éléments
  - Bijection p sur un ensemble fini
  - Représentation sur deux lignes : matrice

$$\begin{pmatrix} 0 & 1 & \dots & i & \dots & n-1 \\ p(0) & p(1) & \dots & p(i) & \dots & p(n-1) \end{pmatrix}$$

  - Représentation fonctionnelle : p codée en C par le tableau `p`

`p[0]`	`p[1]`	...	`p[i]`	...	`p[n-1]`

  - Toute permutation p peut être obtenue par produit de cycles disjoints

Permutation

- Combinatoire : partie des mathématiques discrètes qui étudie des ensembles finis ou dénombrables d'objets
- Combinatoire énumérative : algorithmes pour
  - générer ces objets
  - compter ces objets
- Exemple de structure combinatoire : Permutations de n éléments
  - Bijection p sur un ensemble fini
  - Représentation sur deux lignes : matrice

$$\begin{pmatrix} 0 & 1 & \dots & i & \dots & n-1 \\ p(0) & p(1) & \dots & p(i) & \dots & p(n-1) \end{pmatrix}$$

  - Représentation fonctionnelle : p codée en C par le tableau `p`

`p[0]`	`p[1]`	...	`p[i]`	...	`p[n-1]`

  - Toute permutation p peut être obtenue par produit de cycles disjoints
  - Exemple : $p = (0\ 1\ 3)\ (2\ 5)\ (4) = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 1 & 3 & 5 & 0 & 4 & 2 \end{pmatrix}$

    $\rightsquigarrow$ p codée en C par le tableau

1	3	5	0	4	2

Génération exhaustive

Exemple : permutations de 3 éléments

	cycles	matrice	tableau
1	(0) (1) (2)	$\begin{pmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \end{pmatrix}$	0 1 2
2	(0) (1 2)	$\begin{pmatrix} 0 & 1 & 2 \\ 0 & 2 & 1 \end{pmatrix}$	0 2 1
3	(0 1) (2)	$\begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 2 \end{pmatrix}$	1 0 2
4	(0 1 2)	$\begin{pmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \end{pmatrix}$	1 2 0
5	(0 2 1)	$\begin{pmatrix} 0 & 1 & 2 \\ 2 & 0 & 1 \end{pmatrix}$	2 0 1
6	(0 2) (1)	$\begin{pmatrix} 0 & 1 & 2 \\ 2 & 1 & 0 \end{pmatrix}$	2 1 0

Bibliothèque de générateurs

- Génération de diverses structures combinatoires, encodables en C par un tableau d'entiers structuré
- Tableaux satisfaisant des contraintes structurelles exprimées en logique du premier ordre

Bibliothèque de générateurs

- Génération de diverses structures combinatoires, encodables en C par un tableau d'entiers structuré
- Tableaux satisfaisant des contraintes structurelles exprimées en logique du premier ordre
- Notion de générateur séquentiel
  - Deux fonctions C, générant tous les tableaux d'une taille donnée, l'un après l'autre, dans un ordre total
  - int first_x(int a[], int n,...) génère le premier tableau a de taille n de la famille x
  - int next_x(int a[], int n, ...) génère dans le tableau a de taille n le prochain élément de la famille x, suivant immédiatement celui stocké dans le tableau a

Bibliothèque de générateurs

- Génération de diverses structures combinatoires, encodables en C par un tableau d'entiers structuré
- Tableaux satisfaisant des contraintes structurelles exprimées en logique du premier ordre
- Notion de générateur séquentiel
  - Deux fonctions C, générant tous les tableaux d'une taille donnée, l'un après l'autre, dans un ordre total
  - int first_x(int a[], int n,...) génère le premier tableau a de taille n de la famille x
  - int next_x(int a[], int n, ...) génère dans le tableau a de taille n le prochain élément de la famille x, suivant immédiatement celui stocké dans le tableau a
- Propriétés attendues
  - Correction : chaque tableau généré satisfait ses contraintes structurelles
  - Progression : chaque tableau généré est supérieur au précédent
  - Exhaustivité : tous les tableaux sont générés

Exemple de générateur séquentiel

Générateur des permutations de n éléments
Permutations stockées dans un tableau a de longueur n

```
int first_perm(int a[], int n) {
 for (int i = 0; i < n; i++) a[i] = i;
 return 1;
}
```

Exemple de générateur séquentiel

Générateur des permutations de n éléments
Permutations stockées dans un tableau a de longueur n

```
int first_perm(int a[], int n) {
 for (int i = 0; i < n; i++) a[i] = i;
 return 1;
}
```

0	1	2

Exemple de générateur séquentiel

Générateur des permutations de n éléments
Permutations stockées dans un tableau a de longueur n

0	1	2

```
int first_perm(int a[], int n) {
 for (int i = 0; i < n; i++) a[i] = i;
 return 1;
}
```

```
int next_perm(int a[], int n) {
 ...
 if (...)  return 0;
 ...
 return 1;
}
```

Exemple de générateur séquentiel

Générateur des permutations de n éléments
Permutations stockées dans un tableau a de longueur n

```
int first_perm(int a[], int n) {
 for (int i = 0; i < n; i++) a[i] = i;
 return 1;
}
```

0	1	2

0	2	1

```
int next_perm(int a[], int n) {
 ...
 if (...)  return 0;
 ...
 return 1;
}
```

Exemple de générateur séquentiel

Générateur des permutations de n éléments
Permutations stockées dans un tableau a de longueur n

```
int first_perm(int a[], int n) {
 for (int i = 0; i < n; i++) a[i] = i;
 return 1;
}
```

0	1	2

0	2	1

1	0	2

```
int next_perm(int a[], int n) {
 ...
 if (...)  return 0;
 ...
 return 1;
}
```

Exemple de générateur séquentiel

Générateur des permutations de n éléments
Permutations stockées dans un tableau a de longueur n

```
int first_perm(int a[], int n) {
 for (int i = 0; i < n; i++) a[i] = i;
 return 1;
}
```

0	1	2

0	2	1

1	0	2

1	2	0

```
int next_perm(int a[], int n) {
 ...
 if (...)  return 0;
 ...
 return 1;
}
```

Exemple de générateur séquentiel

Générateur des permutations de n éléments
Permutations stockées dans un tableau a de longueur n

```
int first_perm(int a[], int n) {
 for (int i = 0; i < n; i++) a[i] = i;
 return 1;
}
```

0	1	2

0	2	1

1	0	2

1	2	0

2	0	1

```
int next_perm(int a[], int n) {
 ...
 if (...)  return 0;
 ...
 return 1;
}
```

Exemple de générateur séquentiel

Générateur des permutations de n éléments
Permutations stockées dans un tableau a de longueur n

```
int first_perm(int a[], int n) {
 for (int i = 0; i < n; i++) a[i] = i;
 return 1;
}
```

0	1	2

0	2	1

1	0	2

1	2	0

2	0	1

2	1	0

```
int next_perm(int a[], int n) {
 ...
 if (...)  return 0;
 ...
 return 1;
}
```

Exemple de générateur séquentiel

Générateur des permutations de n éléments
Permutations stockées dans un tableau a de longueur n

```
int first_perm(int a[], int n) {
 for (int i = 0; i < n; i++) a[i] = i;
 return 1;
}
```

0	1	2

```
/*@ requires is_perm(a,n);
  ...
   @ ensures is_perm(a,n); */
int next_perm(int a[], int n) {
 ...
 if (...)  return 0;
 ...
 return 1;
}
```

0	2	1

1	0	2

1	2	0

2	0	1

2	1	0

Propriétés

- Propriétés de correction et de progression
  - Comportement des fonctions de génération spécifié formellement
  - Code annoté avec des invariants et variants de boucle
  - Contrats formels automatiquement prouvés

Propriétés

- Propriétés de correction et de progression
  - Comportement des fonctions de génération spécifié formellement
  - Code annoté avec des invariants et variants de boucle
  - Contrats formels automatiquement prouvés
- Propriété d'exhaustivité
  - Il n'existe pas de tableau entre deux tableaux successifs générés
  - Comporte une quantification existentielle sur un tableau
  - Possiblement non déchargée par les prouveurs automatiques actuels
  - Validation par comptage grâce à une génération exhaustive bornée

Aperçu des résultats

Exemple	code C	ACSL	buts	Durée (s)	vitesse (nb/ms)
suffix	9	12	26	2.873	-
filtering	14	33	51	1.230	-
allex	11	28	40	0.557	-
exall	12	27	40	0.545	-
all2	40	28	40	0.577	-
fct	13	26	43	6.858	23 000
subset	13	22	40	6.428	36 000
rgf	13	28	41	8.359	28 000
sorted	13	30	44	8.448	28 000
comb	18	33	46	29.379	35 000
perm	23	29	50	10.778	22 000
invol	16	22	41	10.281	10 000
rgf $\subset$ endofct	25	27	69	1.340	0.04
comb $\subset$ fct	21	28	67	3.863	0.03
inj $\subset$ fct	29	42	91	1.842	295
surj $\subset$ fct	29	40	103	1.723	230
perm $\subset$ fct	30	42	91	1.493	5
invol $\subset$ perm	20	27	66	1.458	0.04
derang $\subset$ perm	20	27	66	1.440	2.3
ffi $\subset$ invol	20	27	64	1.467	0.3

Contributions

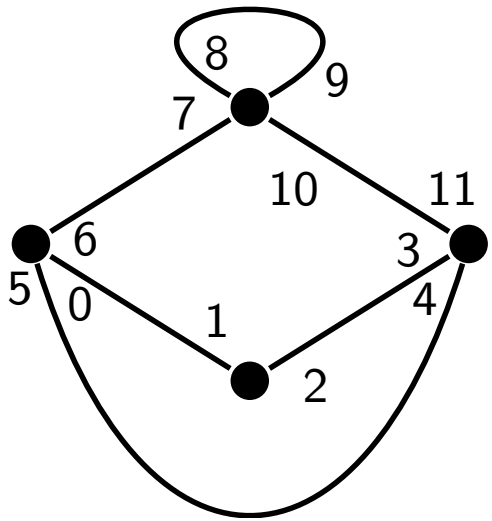
- Patrons généraux d'aide à l'écriture de programmes spécifiés
- Bibliothèque : archive `enum.*.tar.gz` disponible à l'adresse `http://members.femto-st.fr/richard-genestier/en`
- Deux publications

R. Genestier, A. Giorgetti, and G. Petiot.
Sequential generation of structured arrays and its deductive verification.
In *TAP 2015*, volume 9154 of *LNCS*, pages 109–128. Springer, Heidelberg, 2015.

Richard Genestier, Alain Giorgetti, and Guillaume Petiot.
Gagnez sur tous les tableaux.
In David Baelde and Jade Alglave, editors, *Vingt-sixièmes Journées Francophones des Langages Applicatifs (JFLA 2015)*, Le Val d'Ajol, France, January 2015.

Cartes

Exemple de carte combinatoire étiquetée



Carte topologique

$$
\begin{aligned}
D &= \{0,1,2,3,4,5,6,7,8,9,10,11\} \\
L &= (0\ 1)\ (2\ 3)\ (4\ 5)\ (6\ 7)\ (8\ 9)\ (10\ 11) \\
R &= (5\ 0\ 6)\ (1\ 2)\ (11\ 3\ 4)\ (8\ 7\ 10\ 9)
\end{aligned}
$$

Carte combinatoire (D, R, L)

Transitivité: $11 \underset{\rightarrow}{L}\ 10 \underset{\rightarrow}{R}\ 9 \underset{\rightarrow}{R}\ 8 \underset{\rightarrow}{R}\ 7 \underset{\rightarrow}{L}\ 6$

Principe général

Carte combinatoire étiquetée

Triplet (D, R, L)

Principe général

Carte combinatoire étiquetée

Triplet (D, R, L)

- D : ensemble fini à $2n$ éléments

Principe général

Carte combinatoire étiquetée

Triplet (D, R, L)

- D : ensemble fini à $2n$ éléments

- $D = \{0, \ldots, 2n-1\}$

Principe général

Carte combinatoire étiquetée

Triplet (D, R, L)

- D : ensemble fini à $2n$ éléments
- R : permutation sur D

- $D = \{0, \ldots, 2n-1\}$
- Générateur séquentiel de permutations : `first_perm` + `next_perm`

Principe général

Carte combinatoire étiquetée

Triplet (D, R, L)

- D : ensemble fini à $2n$ éléments
- R : permutation sur D
- L : involution sans point fixe sur D

- $D = \{0, \ldots, 2n-1\}$
- Générateur séquentiel de permutations : `first_perm` + `next_perm`
- Générateur séquentiel d'involutions sans point fixe : `first_ffi` + `next_ffi`

Principe général

Carte combinatoire étiquetée

Triplet (D, R, L)

- D : ensemble fini à $2n$ éléments
- R : permutation sur D
- L : involution sans point fixe sur D

- $D = \{0, \ldots, 2n-1\}$
- Générateur séquentiel de permutations : `first_perm` + `next_perm`
- Générateur séquentiel d'involutions sans point fixe : `first_ffi` + `next_ffi`
- Générateur générique de couples de tableaux : `first_pair` + `next_pair`

Principe général

Carte combinatoire étiquetée

Triplet (D, R, L)

- D : ensemble fini à $2n$ éléments
- R : permutation sur D
- L : involution sans point fixe sur D
- Le groupe $\langle R, L \rangle$ agit transitivement sur D

- $D = \{0, \ldots, 2n-1\}$
- Générateur séquentiel de permutations : `first_perm` + `next_perm`
- Générateur séquentiel d'involutions sans point fixe : `first_ffi` + `next_ffi`
- Générateur générique de couples de tableaux : `first_pair` + `next_pair`
- Action transitive caractérisée par une fonction `b_trans`

Principe général

Carte combinatoire étiquetée

Triplet (D, R, L)

- D : ensemble fini à $2n$ éléments
- R : permutation sur D
- L : involution sans point fixe sur D
- Le groupe $\langle R, L \rangle$ agit transitivement sur D

- $D = \{0, \ldots, 2n-1\}$
- Générateur séquentiel de permutations : `first_perm` + `next_perm`
- Générateur séquentiel d'involutions sans point fixe : `first_ffi` + `next_ffi`
- Générateur générique de couples de tableaux : `first_pair` + `next_pair`
- Action transitive caractérisée par une fonction `b_trans`
- Générateur générique de couples transitifs par filtrage utilisant `b_trans` : `first_trans_pair` + `next_trans_pair`

Principe général

Carte combinatoire étiquetée

Triplet (D, R, L)

- D : ensemble fini à $2n$ éléments
- R : permutation sur D
- L : involution sans point fixe sur D
- Le groupe $\langle R, L \rangle$ agit transitivement sur D

$\rightsquigarrow$ Génération exhaustive de cartes étiquetées

- $D = \{0, \ldots, 2n-1\}$
- Générateur séquentiel de permutations : `first_perm` + `next_perm`
- Générateur séquentiel d'involutions sans point fixe : `first_ffi` + `next_ffi`
- Générateur générique de couples de tableaux : `first_pair` + `next_pair`
- Action transitive caractérisée par une fonction `b_trans`
- Générateur générique de couples transitifs par filtrage utilisant `b_trans` : `first_trans_pair` + `next_trans_pair`

Générateur de couples de tableaux

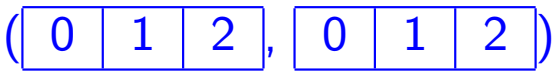
Premier couple

```
int first_pair(int a[], int b[], int n,
 int (*first_x)(int a[], int n),
 int (*first_y)(int b[], int n) ) {

 return (*first_x)(a,n) && (*first_y)(b,n);
}
```

Générateur de couples de tableaux

Premier couple

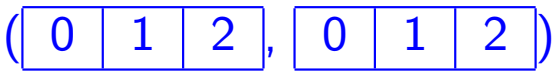
```
int first_pair(int a[], int b[], int n,
 int (*first_x)(int a[], int n),
 int (*first_y)(int b[], int n) ) {

 return (*first_x)(a,n) && (*first_y)(b,n);
}
```



Générateur de couples de tableaux

Premier couple

```
int first_pair(int a[], int b[], int n,
 int (*first_x)(int a[], int n),
 int (*first_y)(int b[], int n) ) {

 return (*first_x)(a,n) && (*first_y)(b,n);
}
```



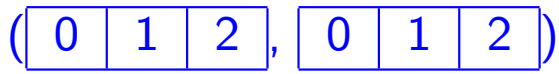
Couples suivants selon l'ordre lexicographique

```
int next_pair(int a[], int b[], int n,
 int (*next_x)(int a[], int n),
 int (*first_y)(int b[], int n),
 int (*next_y)(int b[], int n) ) {

 if ((*next_y)(b,n)) return 1;
 return (*next_x)(a,n) && (*first_y)(b,n);
}
```

Générateur de couples de tableaux

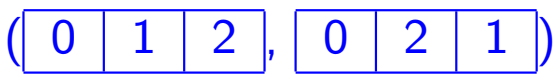
Premier couple

```
int first_pair(int a[], int b[], int n,
 int (*first_x)(int a[], int n),
 int (*first_y)(int b[], int n) ) {

 return (*first_x)(a,n) && (*first_y)(b,n);
}
```



Couples suivants selon l'ordre lexicographique

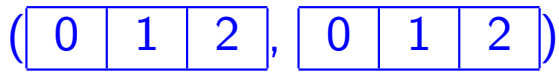
```
int next_pair(int a[], int b[], int n,
 int (*next_x)(int a[], int n),
 int (*first_y)(int b[], int n),
 int (*next_y)(int b[], int n) ) {

 if ((*next_y)(b,n)) return 1;
 return (*next_x)(a,n) && (*first_y)(b,n);
}
```



Générateur de couples de tableaux

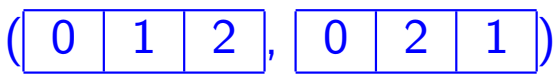
Premier couple

```
int first_pair(int a[], int b[], int n,
 int (*first_x)(int a[], int n),
 int (*first_y)(int b[], int n) ) {

 return (*first_x)(a,n) && (*first_y)(b,n);
}
```

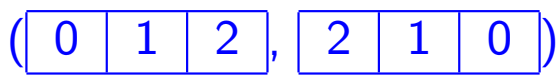


Couples suivants selon l'ordre lexicographique

```
int next_pair(int a[], int b[], int n,
 int (*next_x)(int a[], int n),
 int (*first_y)(int b[], int n),
 int (*next_y)(int b[], int n) ) {

 if ((*next_y)(b,n)) return 1;
 return (*next_x)(a,n) && (*first_y)(b,n);
}
```



...



Générateur de couples de tableaux

Premier couple

```
int first_pair(int a[], int b[], int n,
 int (*first_x)(int a[], int n),
 int (*first_y)(int b[], int n) ) {

 return (*first_x)(a,n) && (*first_y)(b,n);
}
```

(0 1 2, 0 1 2)

Couples suivants selon l'ordre lexicographique

```
int next_pair(int a[], int b[], int n,
 int (*next_x)(int a[], int n),
 int (*first_y)(int b[], int n),
 int (*next_y)(int b[], int n) ) {

 if ((*next_y)(b,n)) return 1;
 return (*next_x)(a,n) && (*first_y)(b,n);
}
```

(0 1 2, 0 2 1)

...

(0 1 2, 2 1 0)

(0 2 1, 0 1 2)

Générateur de couples transitifs

Premier couple transitif

```
int first_trans_pair( int a[], int b[], int n,
 int (*first_x)(int a[], int n),
 int (*next_x)(int a[], int n),
 int (*first_y)(int b[], int n),
 int (*next_y)(int b[], int n) ) {
```

Générateur de couples transitifs

Premier couple transitif

```
int first_trans_pair( int a[], int b[], int n,
 int (*first_x)(int a[], int n),
 int (*next_x)(int a[], int n),
 int (*first_y)(int b[], int n),
 int (*next_y)(int b[], int n) ) {

 int tmp = first_pair(a,b,n,*first_x,*first_y);
 while (tmp != 0 && b_trans(a,b,n) == 0)
  tmp = next_pair(a,b,n,*next_x,*first_y,*next_y);
 if (tmp == 0) return 0;
 return 1;
}
```

Générateur de couples transitifs

Premier couple transitif

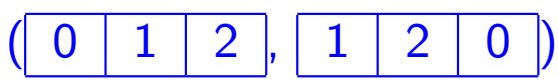
```
int first_trans_pair( int a[], int b[], int n,
 int (*first_x)(int a[], int n),
 int (*next_x)(int a[], int n),
 int (*first_y)(int b[], int n),
 int (*next_y)(int b[], int n) ) {

 int tmp = first_pair(a,b,n,*first_x,*first_y);
 while (tmp != 0 && b_trans(a,b,n) == 0)
  tmp = next_pair(a,b,n,*next_x,*first_y,*next_y);
 if (tmp == 0) return 0;
 return 1;
}
```

Générateur de couples transitifs

Premier couple transitif

```
int first_trans_pair( int a[], int b[], int n,
 int (*first_x)(int a[], int n),
 int (*next_x)(int a[], int n),
 int (*first_y)(int b[], int n),
 int (*next_y)(int b[], int n) ) {

 int tmp = first_pair(a,b,n,*first_x,*first_y);
 while (tmp != 0 && b_trans(a,b,n) == 0)
  tmp = next_pair(a,b,n,*next_x,*first_y,*next_y);
 if (tmp == 0) return 0;
 return 1;
}
```

...



Générateur de couples transitifs

Couples transitifs suivants selon l'ordre lexicographique

```
int next_trans_pair(int a[], int b[], int n,
 int (*next_x)(int a[], int n),
 int (*first_y)(int b[], int n),
 int (*next_y)(int b[], int n)) {
```

Générateur de couples transitifs

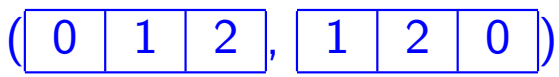
Couples transitifs suivants selon l'ordre lexicographique

```
int next_trans_pair(int a[], int b[], int n,
 int (*next_x)(int a[], int n),
 int (*first_y)(int b[], int n),
 int (*next_y)(int b[], int n)) {

 int tmp;
 do tmp = next_pair(a,b,n,*next_x,*first_y,*next_y);
 while (tmp != 0 && b_trans(a,b,n) == 0);
 if (tmp == 0) return 0;
 return 1;
}
```

Générateur de couples transitifs

Couples transitifs suivants selon l'ordre lexicographique

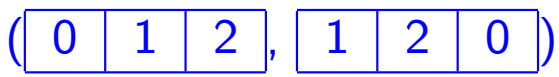
```
int next_trans_pair(int a[], int b[], int n,
 int (*next_x)(int a[], int n),
 int (*first_y)(int b[], int n),
 int (*next_y)(int b[], int n)) {

 int tmp;
 do tmp = next_pair(a,b,n,*next_x,*first_y,*next_y);
 while (tmp != 0 && b_trans(a,b,n) == 0);
 if (tmp == 0) return 0;
 return 1;
}
```

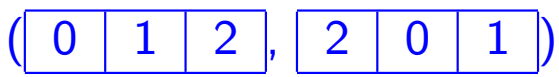


Générateur de couples transitifs

Couples transitifs suivants selon l'ordre lexicographique

```
int next_trans_pair(int a[], int b[], int n,
 int (*next_x)(int a[], int n),
 int (*first_y)(int b[], int n),
 int (*next_y)(int b[], int n)) {

 int tmp;
 do tmp = next_pair(a,b,n,*next_x,*first_y,*next_y);
 while (tmp != 0 && b_trans(a,b,n) == 0);
 if (tmp == 0) return 0;
 return 1;
}
```





Génération exhaustive de cartes

Première carte : appel de

```
first_trans_pair(a,b,2*n,
                 (& first_perm),(& next_perm),
                 (& first_ffi),(& next_ffi));
```

Génération exhaustive de cartes

Première carte : appel de

```
first_trans_pair(a,b,2*n,
                 (& first_perm),(& next_perm),
                 (& first_ffi),(& next_ffi));
```

Cartes suivantes : appels

```
while(next_trans_pair(a,b,2*n,
                      (& next_perm),
                      (& first_ffi),(& next_ffi)))
```

Génération exhaustive de cartes

Première carte : appel de

```
first_trans_pair(a,b,2*n,
                 (& first_perm),(& next_perm),
                 (& first_ffi),(& next_ffi));
```

Cartes suivantes : appels

```
while(next_trans_pair(a,b,2*n,
                      (& next_perm),
                      (& first_ffi),(& next_ffi)))
```

- Adaptable à
  - D'autres types de cartes
  - D'autres fonctions de filtrage
- Utilise les générateurs de la bibliothèque
- Génération efficace
  - Conception d'un générateur efficace d'involutions sans point fixe
- Générateurs non vérifiés
  - Transitivité : quantification existentielle sur un tableau
  - Généricité
- Validation par génération exhaustive bornée

Autres travaux

Génération arborescente

- Deux opérations de construction de cartes
  $\rightsquigarrow$ deux opérations sur les permutations
- Preuve interactive (en Coq) de la transitivité et de la préservation des permutations

Génération arborescente

- Deux opérations de construction de cartes
  $\rightsquigarrow$ deux opérations sur les permutations
- Preuve interactive (en Coq) de la transitivité et de la préservation des permutations
- Deux publications


C. Dubois, A. Giorgetti, and R. Genestier.
Tests and proofs for enumerative combinatorics.
In B. Aichernig and C. A. Furia, editors, *Tests and Proofs (TAP)*, volume 9762 of *LNCS*. Springer, Heidelberg, 2016.


R. Genestier and A. Giorgetti.
Spécification et vérification formelle d'opérations sur les permutations.
In *Approches Formelles dans l'Assistance au Développement Logiciel (AFADL)*, 2016.

Cartes planaires

- Carte planaire (non étiquetée) enracinée à $2n$ brins représentée par un mot à $2n$ lettres issues d'un alphabet à 4 lettres
- Définition d'une relation d'équivalence sur ces mots

Motif π	$a_n, 0 \leq n \leq 9$	Fonction génératrice	OEIS
$\{a\},\{\bar{a}\},\{b\},\{\bar{b}\}$	$1,2,6,20,70,252,924,3432,12870,48620$	$\frac{1}{\sqrt{1-4x}}$	$A000984$
$\{a\bar{a}\},\{b\bar{b}\}$	$1,2,5,13,34,89,233,610,1597,4181$	$\frac{1-x}{1-3x+x^2}$	$A122367$
$\{\bar{a}a\},\{\bar{b}b\}$	$1,1,2,5,13,34,89,233,610,1597$	$\frac{1-2x}{1-3x+x^2}$	$A001519$
$\{aa\},\{\bar{a}\bar{a}\},\{bb\},\{\bar{b}\bar{b}\}$	$1,1,2,5,12,31,81,216,583,1590$	$\frac{1+x-\sqrt{1-2x-3x^2}}{2x^2+3x-1+\sqrt{1-2x-3x^2}}$	Termes de rangs pairs de A191385
$\{ab\},\{ba\},\{a\bar{b}\},\{\bar{b}a\}$	$1,1,2,4,9,20,47,109,262,622$	$\frac{-2\sqrt{3}\sin\left(1/3\ \arcsin\left(3/2\ \sqrt{3}x\right)\right)}{4\left(\sin\left(1/3\ \arcsin\left(3/2\ \sqrt{3}x\right)\right)\right)^2-3\,x}$	$A138164$

J.-L. Baril, R. Genestier, A. Giorgetti, and A. Petrossian.
Rooted planar maps modulo some patterns.
Discrete Mathematics, 339:1199–1205, 2016.

Questions

- Merci pour votre attention
- Questions ?