VERIFICATION DU MODELE DES CARTES COMBINATOIRES ORIENTEES A L'AIDE DE FRAMA-C

Projet Galapagos

C. Dubois, Y. Harbit, N. Magaud

Objectif

- Vérification d'une implantation en C du modèle des cartes combinatoires et de ses opérations de base
- Utilisation de FramaC (plateforme logicielle de vérifiaction de programes C) développée au CEA.
  Plus particulièrement plugin Jessie de vérification déductive.
- Jusqu'à présent, preuves en Coq, en B sur les cartes et G-cartes, pas de vérification de code

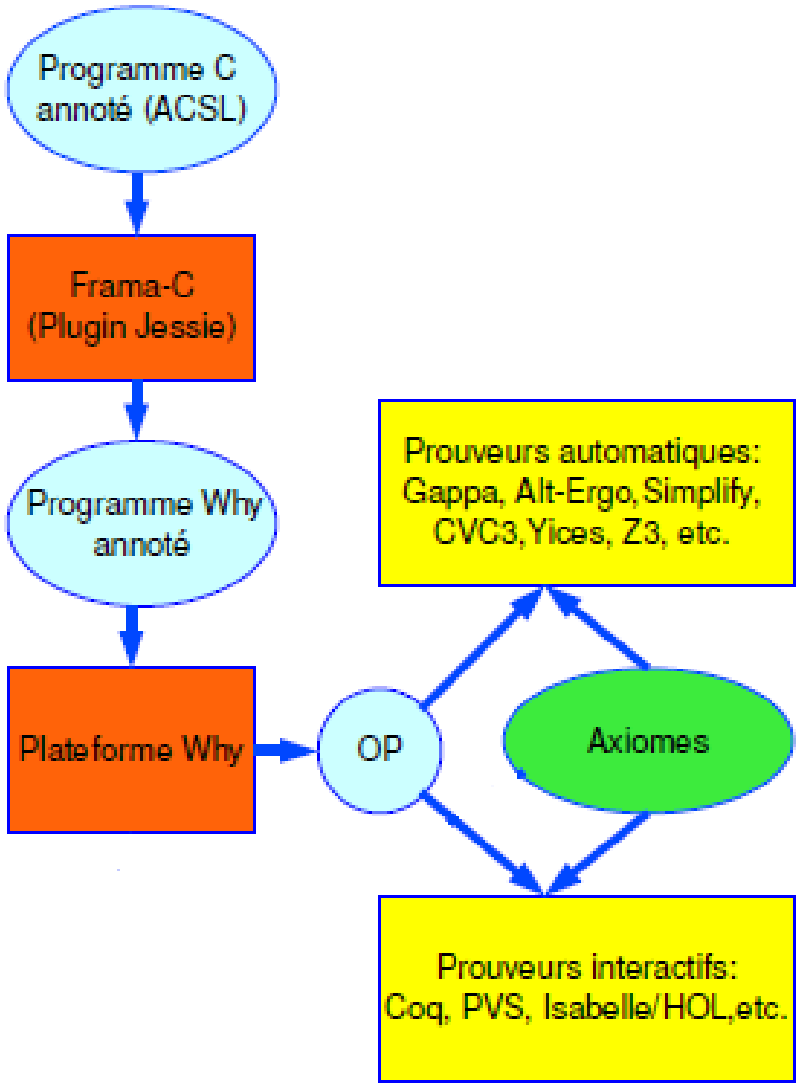
Quelques mots sur Jessie

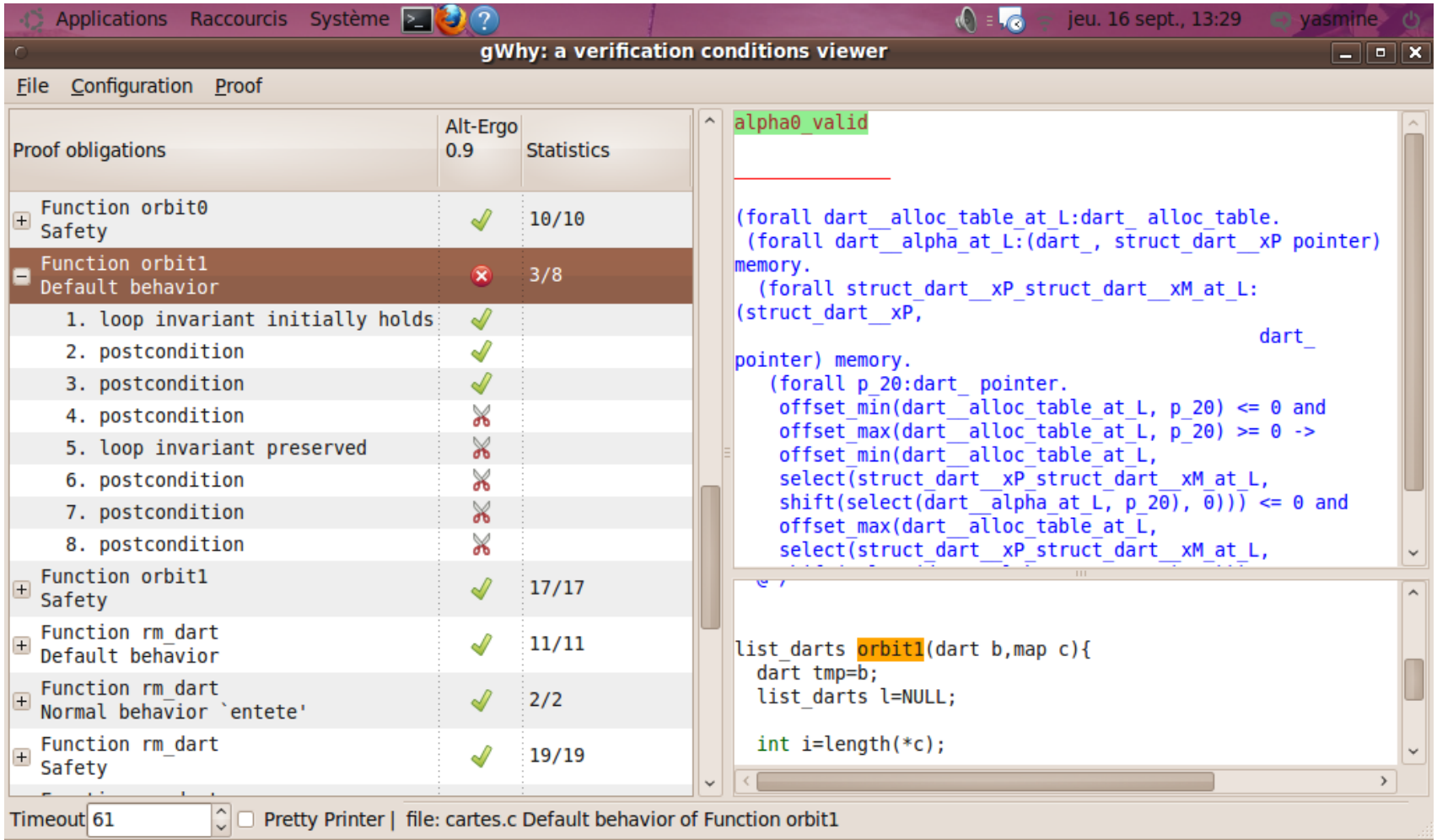
Programmes C + annotations ACSL

- ACSL (Ansi C specification language), langage permettant de spécifier le comportement des programmes C (inspiré de JML, langage d'annotations pour les programmes Java)
- Langage du 1er ordre, syntaxe à la C
- Permet d'écrire des contrats (requires, ensures), de mettre des assertions dans le code (assert)
- Correction totale : invariant, variant dans les boucles

$\Rightarrow$ Génération d'obligations de preuve

Jessie utilise Why, générateur d'obligations de preuve (J.C Filliatre) (Jessie *successeur* de Caduceus)



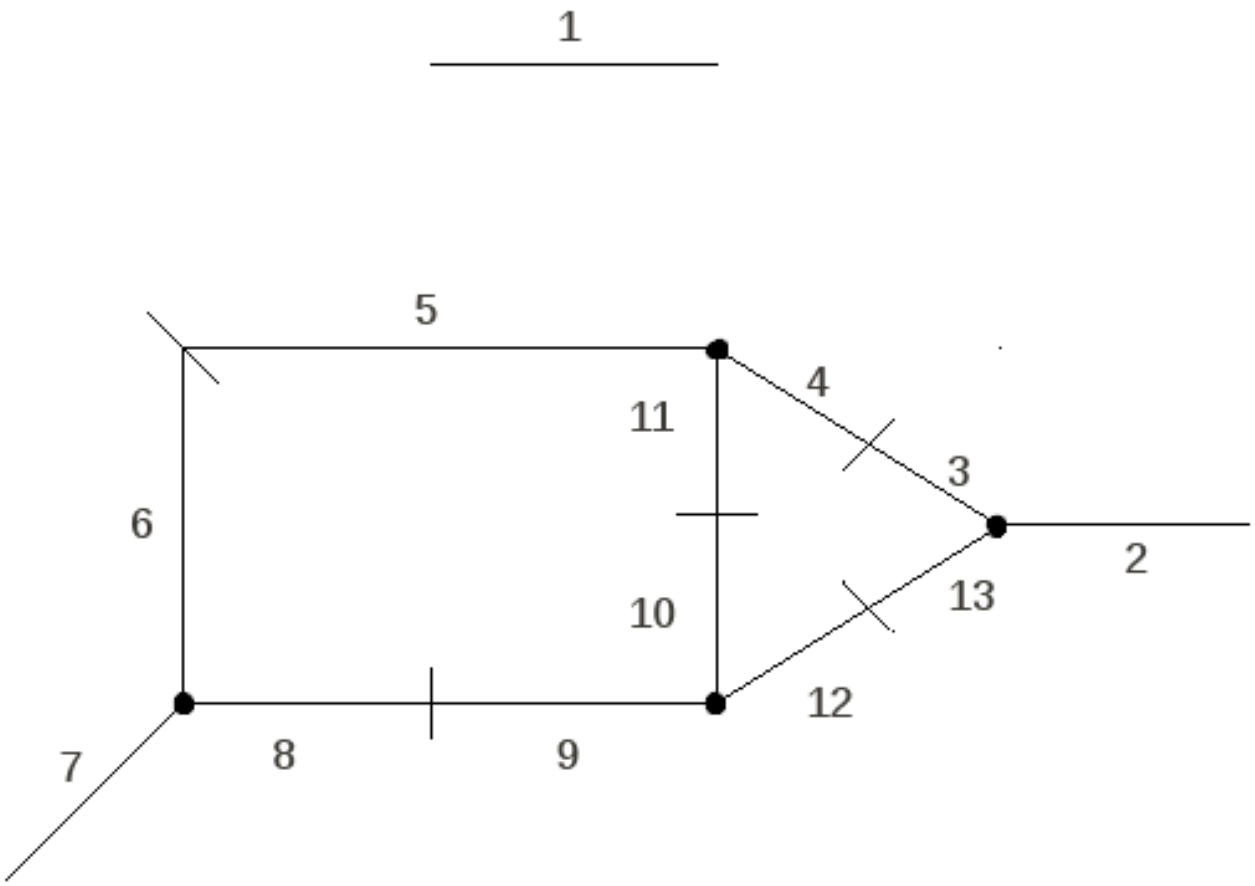

Applications Raccourcis Système
jeu. 16 sept., 13:29 yasmine
gWhy: a verification conditions viewer
File Configuration Proof
Proof obligations
Alt-Ergo 0.9
Statistics
Function orbit0 Safety
10/10
Function orbit1 Default behavior
3/8
1. loop invariant initially holds
2. postcondition
3. postcondition
4. postcondition
5. loop invariant preserved
6. postcondition
7. postcondition
8. postcondition
Function orbit1 Safety
17/17
Function rm_dart Default behavior
11/11
Function rm_dart Normal behavior `entete'
2/2
Function rm_dart Safety
19/19
alpha0_valid
(forall dart__alloc_table_at_L:dart_ alloc_table.
(forall dart__alpha_at_L:(dart_, struct_dart__xP pointer)
memory.
(forall struct_dart__xP_struct_dart__xM_at_L:
(struct_dart__xP,
dart_
pointer) memory.
(forall p_20:dart_ pointer.
offset_min(dart__alloc_table_at_L, p_20) <= 0 and
offset_max(dart__alloc_table_at_L, p_20) >= 0 ->
offset_min(dart__alloc_table_at_L,
select(struct_dart__xP_struct_dart__xM_at_L,
shift(select(dart__alpha_at_L, p_20), 0))) <= 0 and
offset_max(dart__alloc_table_at_L,
select(struct_dart__xP_struct_dart__xM_at_L,
list_darts orbit1(dart b,map c){
dart tmp=b;
list_darts l=NULL;
int i=length(*c);
Timeout 61
Pretty Printer | file: cartes.c Default behavior of Function orbit1

Modèle des cartes combinatoires orientées

Une carte combinatoire (orientée) $= (D, \alpha_0, \alpha_1)$ avec

- D un ensemble de brins
- α_0 une involution sur D
- α_1 une permutation sur D

- On dit que b_1 est cousu à b_2 à la dimension i si $\alpha_i(b_1) = b_2$
- Un brin est i-libre si $\alpha_i(b) = b$



D	1	2	3	4	5	6	7	8	9	10	11	12	13
α_0	1	2	4	3	6	5	7	9	8	11	10	13	14
α_1	1	3	13	5	11	7	8	6	12	9	4	10	2

Les opérations de base

- Créer une carte vide
- Ajouter un brin
  Le brin est i-libre (pour i=0,1) dans la nouvelle carte
- Coudre 2 brins à une dimension donnée
  Les deux brins doivent être dans la carte et i-libres.
- Découdre un brin à une dimension donnée
  Le brin doit être cousu à la dimension donnée
  Le brin et son image deviennent i-libres
- Retirer un brin
  le brin doit être i-libre

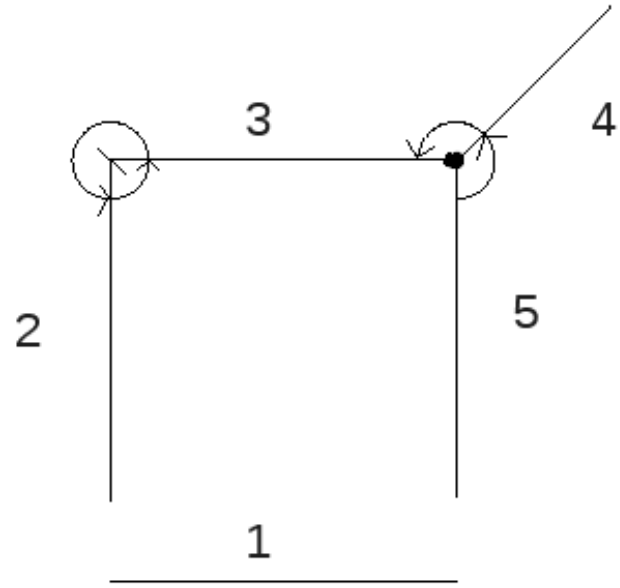
Implantation en C

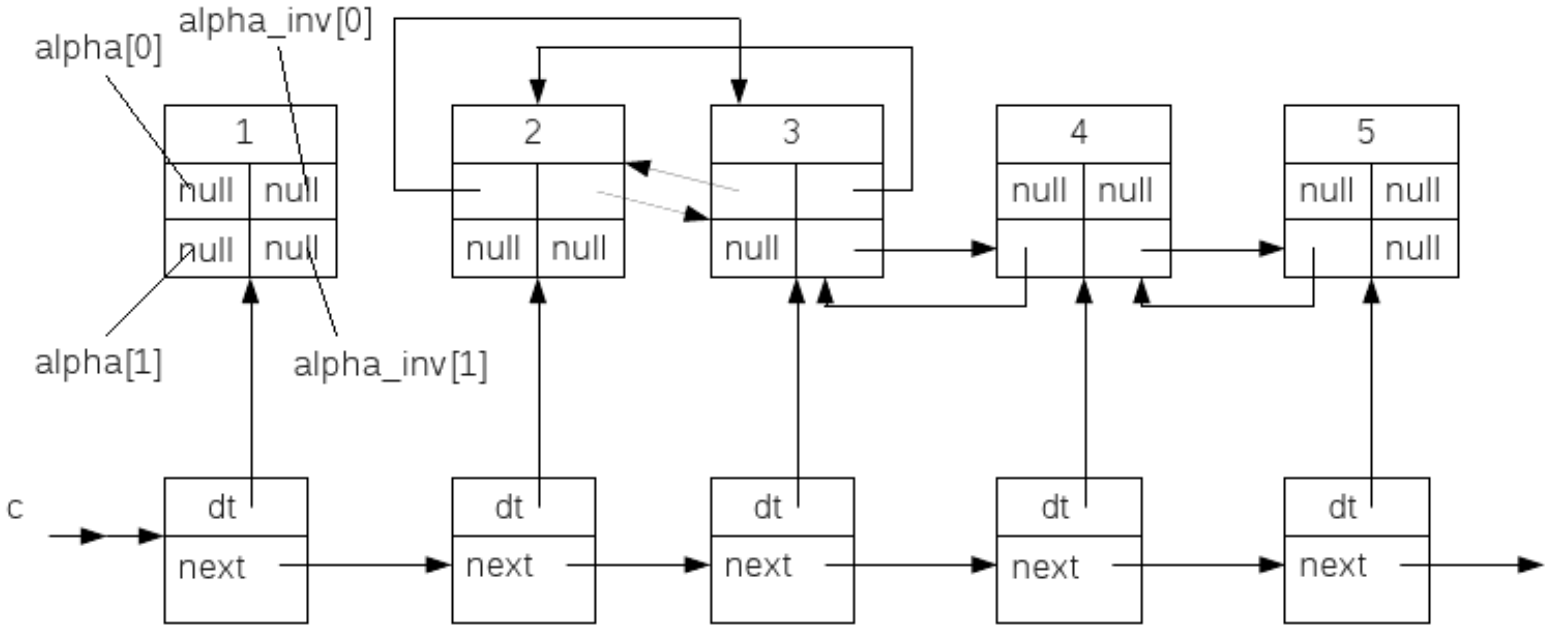
```
 struct dart_ {
  int id;
  struct dart_* alpha [2];
  struct dart_* alpha_inv [2];
};

typedef struct dart_ *dart;

struct list_darts_ {
  dart dt;
  struct list_darts_ *next;};

typedef struct list_darts_ ** map;
typedef struct list_darts_ * list_darts;
```

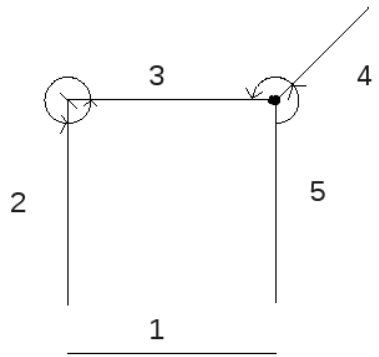

3
4
2
5
1

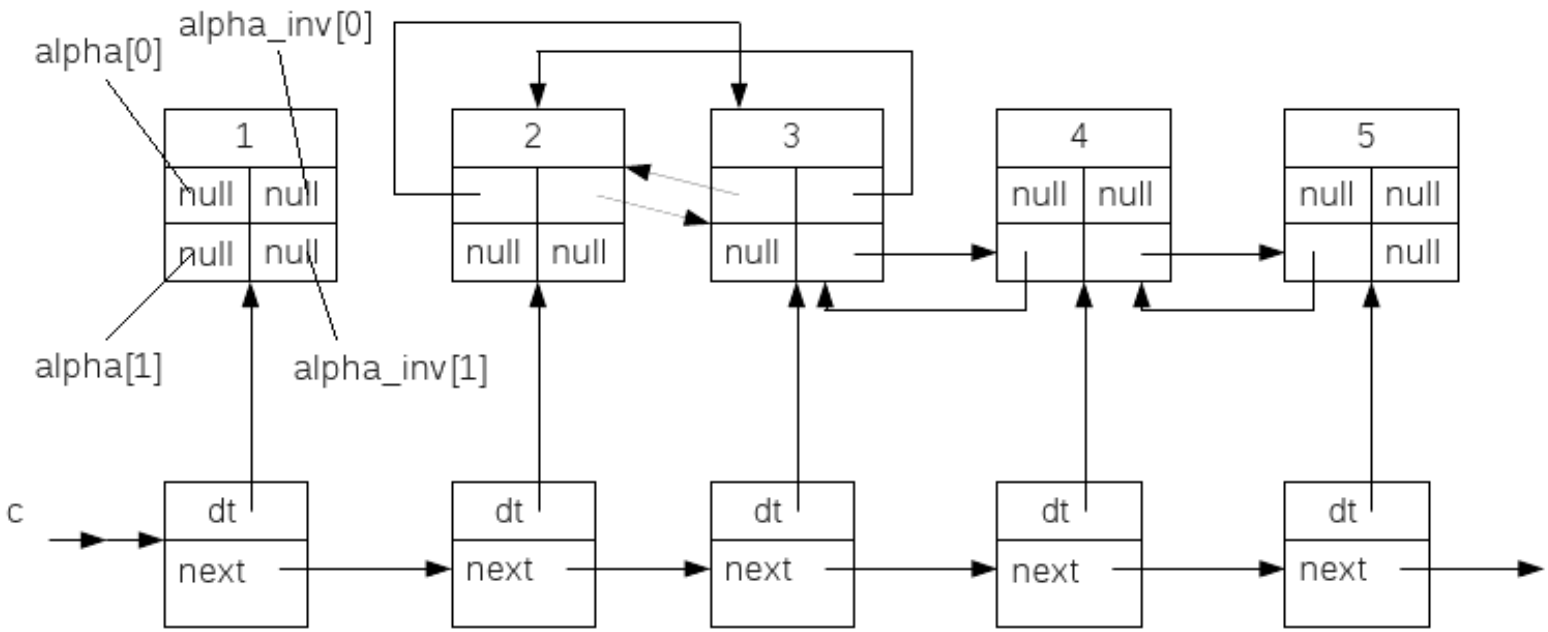

alpha[0]
alpha_inv[0]
1
2
3
4
5
null
null
null
null
null
null
null
null
null
null
null
null
null
null
alpha[1]
alpha_inv[1]
c
dt
next
dt
next
dt
next
dt
next
dt
next

Choix d'implantation

Un brin i-libre a ses pointeurs alpha[i] et alpha_inv[i] à NULL

Les 1-orbites sont représentées *ouvertes*





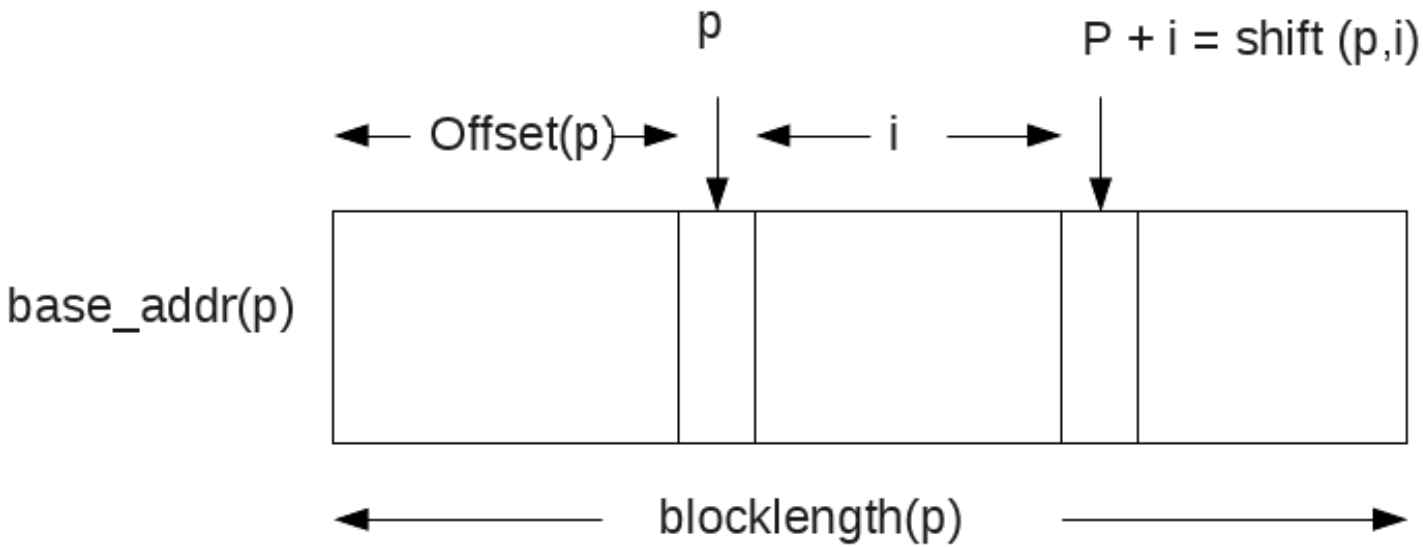
Les vérifications de Jessie

- vérifications de sûreté (safety checks)
  - Mémoire : validité des accès à la mémoire
  - Entiers : absence d'overflows, validité des opérations sur les entiers (absence de division par 0)
- vérifications fonctionnelles
  - invariants de représentation (par ex. alpha[i] et alpha_inv[i] cohérents)
  - invariants du modèle (α_0 involution, α_1 permutation)
  - comportement des opérations

Vérification 'Safety'

Modèle mémoire

Un pointeur p est représenté par une paire (adresse de base, décalage dans le bloc)



$$\text{valid}\ (p) \equiv p \neq \text{null} \wedge 0 \leq \text{offset}(p) < \text{blocklength}(p)$$

Annotations

Un peu partout ajout d'annotations concernant la validité des pointeurs (carte, brins) en paramètre ou en résultat des opérations.
Insuffisant ! tous les brins d'une carte sont valides.
Définition d'un prédicat inductif : dt_valid

```
/*@ inductive dt_valid{L} (list_darts p1) {
  @ case dt_null{L}:dt_valid(NULL);
  @ case dt_next{L}:
  @ \forall list_darts p;
  @ \valid(p) ==> \valid(p->dt) ==> dt_valid(p->next)
  @ ==> dt_valid(p);} @*/
```

+ quelques lemmes

```
/*@ requires \valid(b) && \valid(c) && \valid(*c) &&
 @  dt_valid(*c);
 @*/
int exd (dart b,map c) {
  list_darts tmp=(*c);
  /*@ loop invariant dt_valid(tmp)
  @*/
  while (tmp!=NULL){
    if (tmp->dt->id==b->id) {
      return 1;}
  tmp=tmp->next;
  }
  return 0;
}
```

⇒ Pour l'ensemble du code : 88 obligations de preuve concernant le déréférencement de pointeurs, toutes prouvées automatiquement

Terminaison des boucles, overflows

```
/*@ requires \valid(b) && \valid(c) && \valid(*c) &&
 @  dt_valid(*c);
 @*/
int exd (dart b,map c) {
  list_darts tmp=(*c);
  /*@ loop invariant dt_valid(tmp)
  @ loop variant list_length(tmp);
  @*/
  while (tmp!=NULL){
    if (tmp->dt->id==b->id) {
      return 1;}
  tmp=tmp->next;
  }
  return 0;
}
```

avec list_length spécifié en ACSL

⇒ Pour l'ensemble du code : 10 + 12 obligations de preuve, toutes prouvées automatiquement

Invariants

Objectif : préservation de la bonne formation de la carte par les opérations

- null_term_list(l): la liste de brins l est linéaire
- dart_in_map(c) : pour tout brin d'une carte, les brins référencés par alpha[0] et alpha[1] sont dans la carte ou sont NULL.
- dart_inverse(c) : dans la carte, les liens alpha[i] et alpha_inv[i] sont cohérents (`b->alpha[i]->alpha_inv[i]==b`)
- darts_diff(c) : tous les brins de la carte sont différents (ont des identificateurs différents)
- alpha0_inv(c): α_0 est une involution pour la carte c
- propriété sur α_1 non traitée (faire le lien entre permutation et 1-orbites ouvertes)

utilisation d'une notion de liste dans la logique.

```
/*@ requires ....
  @ requires darts_diff(*c) && dart_in_map(c) &&
  @ dart_inverse(c) && alpha0_inv(c) ;
  @ ensures darts_diff(*c) && dart_in_map(c) &&
  @ dart_inverse(c) && alpha0_inv(c) ;
 @*/
int exd (dart b,map c) {
  list_darts tmp=(*c);
  /*@ loop invariant ... && reach(*c,tmp);
  @ loop variant list_length(tmp);
  @*/
  while (tmp!=NULL){
    if (tmp->dt->id==b->id) {
      return 1;}
  tmp=tmp->next;
  }
  return 0;
}
```

⇒ Pour l'ensemble du code : environ 500 obligations de preuve, 7 non prouvées (allocation dynamique)

Conclusion et Perspectives

- Compléter la spécification et la vérification (la plus automatique possible)
- Vérifier la cohérence des spécifications ACSL
- Faire le lien avec les modèles Coq des cartes combinatoires orientées (modèle inductif à la Jean François ? modèle ensembliste à la B ?)