1

Définition et preuve d'un programme fonctionnel de segmentation d'image 2D avec des hypercartes

J.-F. Dufourd *

JFLA'07, Aix-les-Bains, 27-30 janvier 2007

*LSIIT, UMR CNRS-ULP 7005, Pôle Technologique, Boulevard S. Brant, BP10413, F67412, Illkirch, e-mail : dufourd@lsiit.u-strasbg.fr

1 Plan

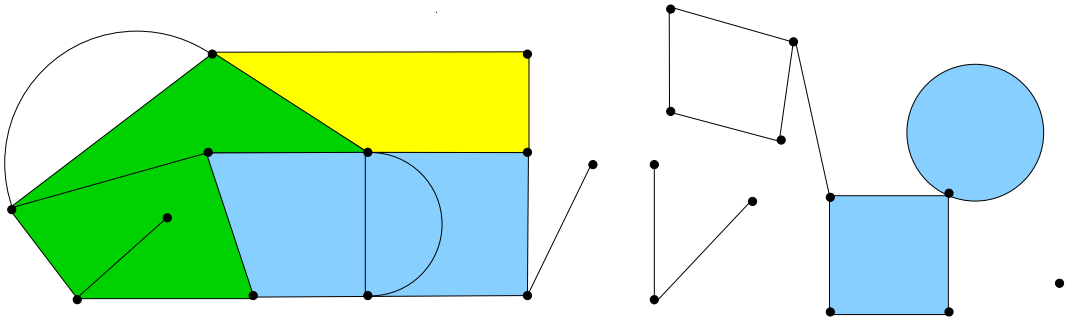
1. Introduction
2. Segmentation et partitions
3. Travaux connexes
4. Aspects mathématiques
5. Spécifications préliminaires
6. Spécification des quasi-hypercartes et hypercartes
7. Spécification de la segmentation en 2 étapes
8. Certification de la segmentation
9. Dérivation d'un programme impératif optimal en C
10. Conclusion

2 Segmentation et partitions

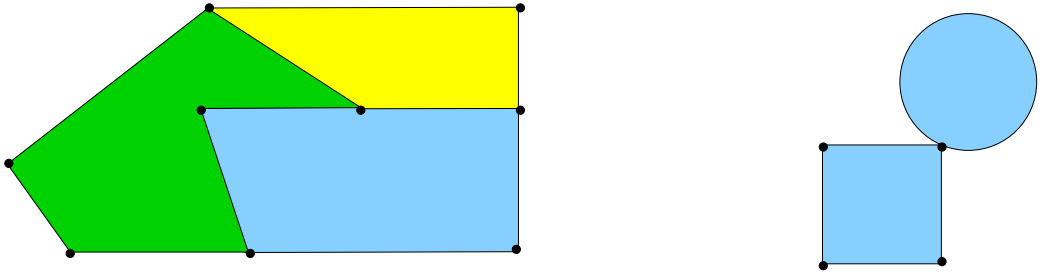
Soit π un plan euclidien support d'une image continue.

• Ordre entre les partitions de π : une partition B est *plus grossière* qu'une partition A si chaque classe de B contient une classe de A.
• Hypothèse : toutes les partitions considérées satisfont un *critère* donné.
• La *segmentation* consiste, à partir d'une partition initiale A, à trouver une partition plus grossière que A et *maximale* pour cet ordre.

• Ici, problème de segmentation très simple opérant sur des partitions du plan en subdivisions finies colorées.
• Critère : préservation des couleurs des points, coloration homogène des cellules, et conservation de tous les sommets et arêtes frontières entre deux faces de couleurs différentes.



a. Une subdivision planaire colorée.



b. Segmentation de la subdivision en (a).

Figure 1: Subdivision planaire finie et sa segmentation.

3 Travaux connexes

• Plus de 1000 algorithmes de segmentation d'images [Serra...]
• Modèles de subdivisions : modèles de cartes combinatoires [Jacques, Tutte..] et extensions [Lienhardt...], hypercartes [Cori...]
• Représentation d'images 2D et 3D avec des modèles de cartes et segmentation [Braquelaire, Brun, Damiand, Domenger, Fioriot...]
• Spécification algébrique des cartes et extensions : modeleur Topofil, coraffinement par réécriture [Bertrand, Cazier, Dufourd]
• Calcul des constructions inductives [Coquant, Huet, Paulin...], assistant de preuves Coq [Equipe Coq INRIA...]
• Certification en algorithmique géométrique avec un assistant de preuves [Bertot, Fleuriot, Meickle, Pichardie]
• Spécifications et preuves en Coq dans les cartes et extensions pour la modélisation des surfaces combinatoires [Dehlinger, Dufourd, Puitg]
• Spécification d'hypercartes et preuve en Coq du théorème des 4 couleurs [Gonthier].

4 Aspects mathématiques

Définition 1 *(*Hypercarte*)*
(i) Une hypercarte *est une structure algébrique* $M = (D, \alpha_0, \alpha_1)$*, où* D *est un ensemble fini de* brins*, et* α_0*,* α_1 *sont des permutations dans* D*.*
(ii) Si $y = \alpha_k(x)$*,* y *est le* k-successeur *de* x*,* x *est le* k-prédécesseur *de* y*, et* x *et* y *sont dits* k-liés*, ou* k-cousus*.*

Définition 2 *(*Orbites et cellules d'hypercarte*)*
(i) Soit D *un ensemble et* $f_1, \ldots, f_n$ *des fonctions dans* D*. L'*orbite *de* $x \in D$ *pour ces fonctions est le sous-ensemble de* D *noté* $< f_1, \ldots, f_n > (x)$ *des éléments accessibles depuis* x *par n'importe quelle composition des fonctions* $f_1, \ldots, f_n$*.*
(ii) Dans l'hypercarte $M = (D, \alpha_0, \alpha_1)$*,* $< \alpha_0 > (x)$ *est la* 0*-orbite ou l'*arête *du brin* x*,* $< \alpha_1 > (x)$ *sa* 1*-orbite ou son* sommet*,* $< \alpha_1^{-1} \circ \alpha_0^{-1} > (x)$ *sa* face*, et* $< \alpha_0, \alpha_1 > (x)$ *sa* composante connexe*. Sommets, arêtes, et faces sont aussi appelés* k-cellules topologiques*, pour* $k = 0, 1$ *et* 2*.*

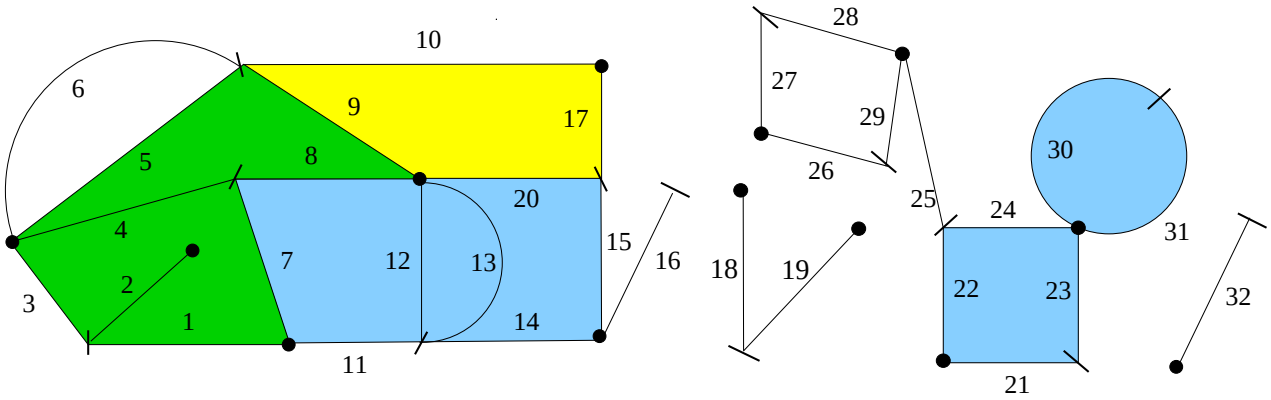


Figure 2: Un exemple d'hypercarte.

• *Plongement* dans un plan : projection des cellules topologiques sur des cellules géométriques de manière à former une subdivision finie du plan : sommets et arêtes sur des points, brins sur des courbes de Jordan ouvertes (éventuellement vides), faces sur des zones connexes.

5 Spécifications de base

• En Coq, dimensions :

```
Inductive dim:Set:= zero: dim | one: dim.
```

Décidabilité de l'égalité = dans `dim`

```
Lemma eq_dim_dec:forall i j:dim,{i=j}+{~i=j}.
```

• Brins :

```
Definition dart:= nat.
Definition eq_dart_dec:= eq_nat_dec.
Definition nil:= 0.
```

• Couleurs (ou niveaux de gris) : `nat`

• *Cartes libres* (ou *free maps*) :

```
Inductive fmap:Set:=
      V : fmap
    | I : fmap->dart->nat->fmap
    | L : fmap->dim->dart->dart->fmap.
```

• *Observateurs* de cartes libres :

- Existence d'un brin dans une carte, et décidabilité `exd_dec` de `exd` :

```
Fixpoint exd(m:fmap)(z:dart){struct m}:Prop:=
 match m with
       V => False
    |  I m0 x _ => z=x \/ exd m0 z
    |  L m0 _ _ _  => exd m0 z
 end.

Lemma exd_dec:forall(m:fmap)(z:dart), {exd m z}+{~exd m z}.
```

```
Lemma exd_dec: forall (m:fmap)(z:dart),
  {exd m z} + {~exd m z}.
Proof.
induction m.
 (* Case V: *)
   right. intro. inversion H.
 (* Case I: *)
   intro. simpl.
   elim (IHm z).
     left. simpl in |- *. tauto.
     intro. elim (eq_dart_dec z d).
        tauto. tauto.
 (* Case L: *)
   intro.elim (IHm z).
     left. simpl in |- *. assumption.
     simpl in |- *.tauto.
Defined.
```

- Opération α_k nommée `A` et existence de successeur `succ` :

```
Fixpoint A(m:fmap)(k:dim)(z:dart)
         {struct m}:dart:=
 match m with
      V => nil
    | I m0 x _ => A m0 k z
    | L m0 k0 x y =>
          if (eq_dim_dec k k0)
          then if (eq_dart_dec z x) then y
               else A m0 k z
          else A m0 k z
 end.

Definition succ(m:fmap)(k:dim)(z:dart):= A m k z <> nil.

Lemma succ_dec:forall(m:fmap)(k:dim)(z:dart),
  {succ m k z} + {~succ m k z}.
```

• *Destructeurs* de cartes libres : suppression de brin `D`, rupture de couture `B` :

```
Fixpoint D(m:fmap)(z:dart){struct m}:map:=...
```

```
Fixpoint B(m:fmap)(k:dim)(z:dart){struct m}:map:=...
```

• *Inverses* : `A_1`, `pred`, `B_1`...

6 Quasi-hypercartes et hypercartes

• *Préconditions* :

```
Definition prec_I(m:fmap)(x:dart):=
  x <> nil /\ ~ exd m x.

Definition prec_L(m:fmap)(k:dim)(x y:dart):=
  exd m x /\ exd m y /\
   ~ succ m k x /\ ~ pred m k y.
```

• *Invariant* des *quasi-hypercartes* :

```
Fixpoint inv_qhmap(m:fmap){struct m}:Prop:=
  match m with
      V => True
    | I m0 x _ => inv_qhmap m0 /\ prec_I m0 x
    | L m0 k x y =>
        inv_qhmap m0 /\ prec_L m0 k x y
  end.
```

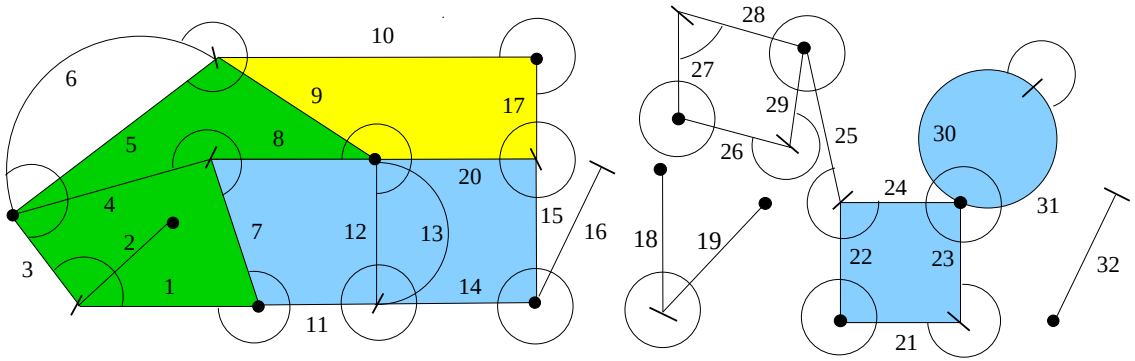


Figure 3: Quasi-hypercarte pour l'hypercarte en exemple.

• *Preuves* : on démontre que, pour toute quasi-hypercarte `m` et dimension `k`, `(A m k)` et `(A_1 m k)` sont des *injections* inverses l'une de l'autre.

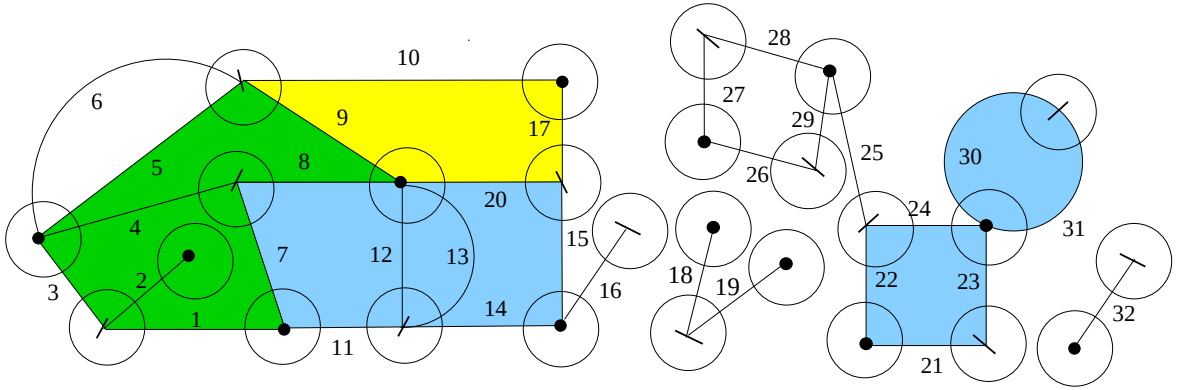


Figure 4: Hypercarte pour la subdivision en exemple.

• Une *hypercarte* est une quasi-hypercarte *complète*, d'invariant `inv_hmap` :

```
Definition inv_hmap(m:fmap):Prop:=
   inv_qhmap m /\
     forall (x:dart)(k:dim),exd m x -> succ m k x /\ pred m k x.
```

• *Preuve* : pour toute hypercarte `m` et toute dimension `k`, `(A m k)` et `(A_1 m k)` sont des *permutations*.

• Existence d'un *chemin dans une face* :

```
expf:  fmap -> dart -> dart -> Prop
```
avec `expf_dec m x y`.

• Théorème du *genre*.

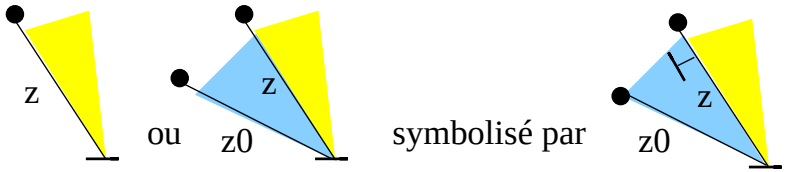
• Formule d'*Euler*.

7 Un algorithme fonctionnel de segmentation

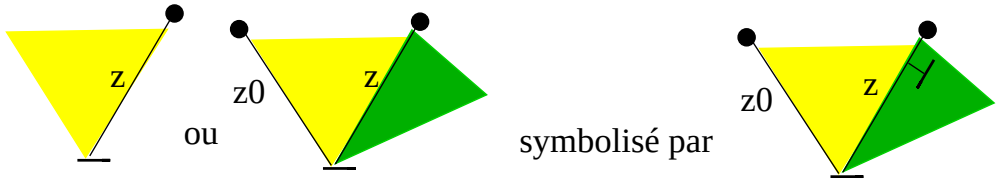
7.1 Première étape de segmentation : seg1

• Notion de brin *frontière* dans une quasi-hypercarte colorée :

```
Definition up_border0(m:fmap)(z:dart)(a:tag):Prop:=
  ~succ m zero z \/ col m (A m zero z) <> a.

Lemma up_border0_dec: forall(m:fmap)(z:dart)(a:tag)
  {up_border0 m z a} + {~up_border0 m z a}

Definition low_border0....

Definition up_border1...

Definition low_border1....
```



a. up_border0 m z (col m z0)



b. low_border0 m z (col m z0).

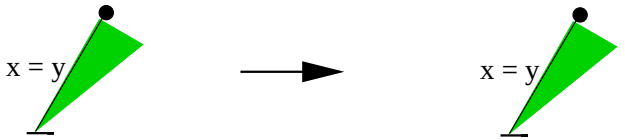
Figure 5: Frontières à la dimension 0.

• Définition fonctionnelle inductive de `seg1 m` :

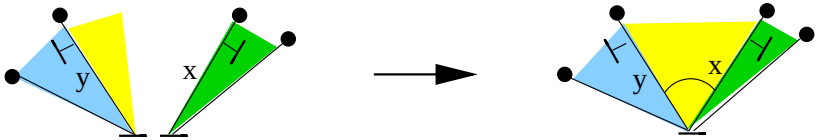
```
Fixpoint seg1(m:fmap):fmap:=
  match m with
      V => V                                          // 1
    | I m0 x c => I (seg1 m0) x c                     // 2
    | L m0 zero x y  =>                               // 3
      let m1 := seg1 m0 in
      let a := (col m0 y) in
      if eq_dart_dec x y then m1                      // 3.1
      else
        let x_0:= A_1 m1 zero x in
        let y0 := A m1 zero y in
        if up_border0_dec m1 y a
        then
         if low_border0_dec m1 x a                    // 3.2
         then L m1 zero x y
         else L (B_1 m1 zero x) zero x_0 y            // 3.3
        else
```

```
        if low_border0_dec m1 x a                    // 3.4
        then L (B m1 zero y) zero x y0
        else                                         // 3.5
         if eq_dart_dec x_0 y                        //3.5.1
         then B m1 zero y
         else                                        //3.5.2
 L (B (B_1 m1 zero x) zero y) zero x_0 y0
  | L m0 one x y  =>                                 //4
    (* ... analogue ... *)
end.
```

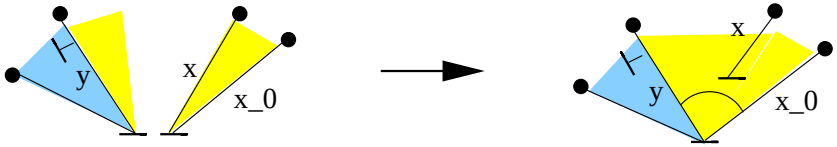
• Maintien d'un *invariant* : jamais 2 faces de même couleur adjacentes cousues par un sommet ou une arête.



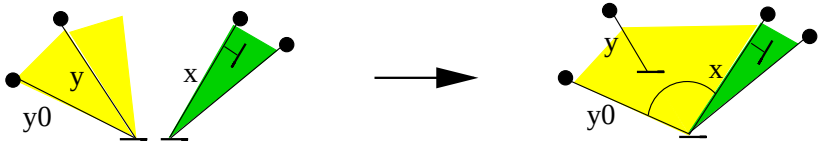
3.1. x = y.



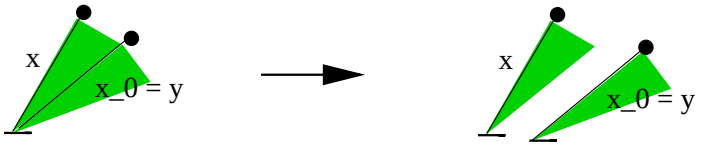
3.2. up_border0 m1 y a /\ low_border0 m1 x a.



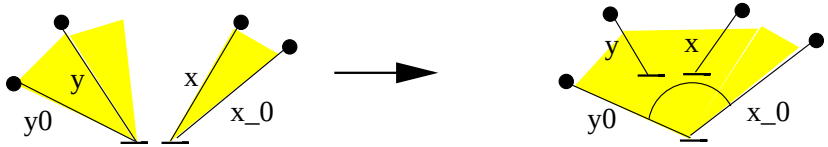
3.3. up_border0 m1 y a /\ ~low_border0 m1 x a.



3.4. ~up_border0 m1 y a /\ low_border0 m1 x a.



3.5.1. ~up_border0 m1 y a /\ ~low_border0 m1 x a /\ x_0 = y.



3.5.2. ~up_border0 m1 y a /\ ~low_border0 m1 x a /\ x_0 <> y.

7.2 Deuxième étape de segmentation : seg2

• Brins *isolés* : `isolated`, avec `isolated_dec` :

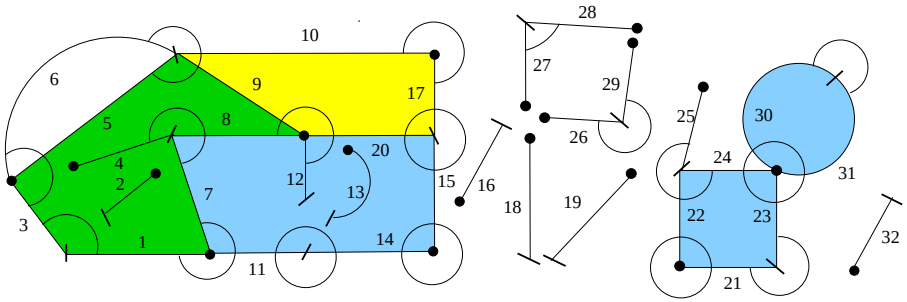
```
Definition isolated(m:fmap)(z:dart):Prop:= ...
```

• Opération auxiliaire `seg2_aux` :

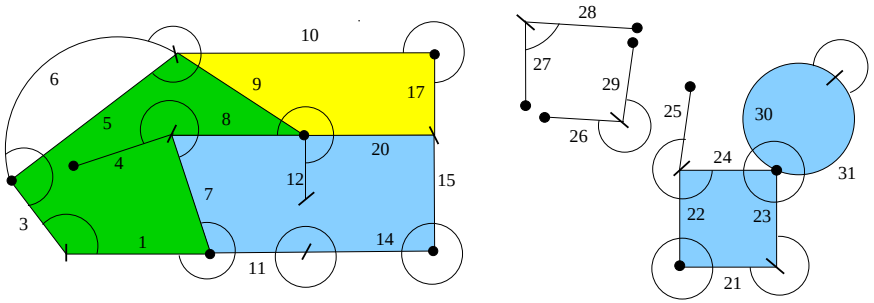
```
Fixpoint seg2_aux(m mr:fmap){struct m}:fmap:=
  match m with
     V => V
   | I m0 x c =>
      if isolated_dec mr x then seg2_aux m0 mr
      else I (seg2_aux m0 mr) x c
   | L m0 k x y =>  L (seg2_aux m0 mr) k x y
  end.
```

• Opérations `seg2` et `segmentation` :

```
Definition seg2(m:fmap):= seg2_aux m m.
Definition segmentation(m:fmap):= seg2 (seg1 m).
```

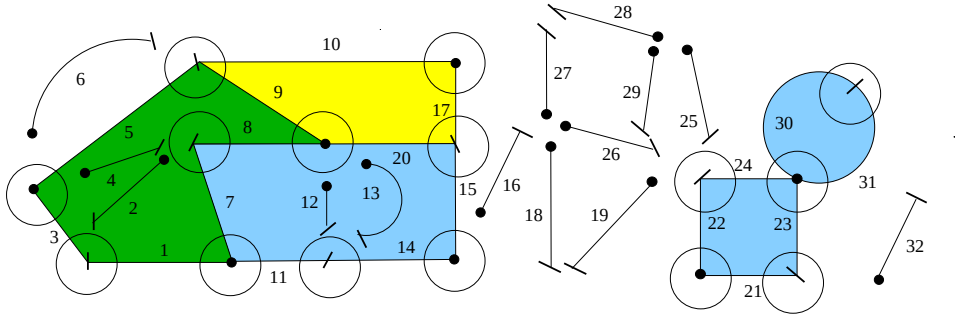


a. Segmentation de m, étape 1 : (seg1 m) .

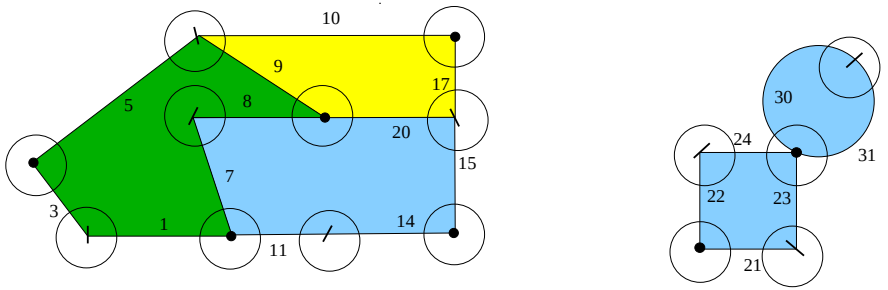


b. Segmentation de m, étapes 1 et 2 : (segmentation m).

Figure 7: Segmentation de la quasi-hypercarte en Fig. 3.



a. Segmentation de m, étape 1 : (seg1 m).



b. Segmentation de m, étapes 1 et 2 : (segmentation m).

Figure 8: Segmentation de l'hypercarte en Fig. 4.

8 Certification de la segmentation

8.1 Correction de la première étape : seg1

• a. *Terminaison finie* de `seg1` : vient de l'induction structurelle.

• b. *Préservation de tous les brins* de `m` avec leurs couleurs.

• c. *Préservation de type* :

```
Theorem inv_qhmap_seg1:forall(m:fmap),
  inv_qhmap m -> inv_qhmap (seg1 m).
```

+ Les brins sans k-successeurs ou k-prédécesseurs le restent.

• d. *Invariant* mentionné en cousant satisfait :

```
Lemma succ0_pred0_seg1:forall(m:fmap)(z:dart),
 inv_qhmap m -> let m1 := seg1 m in
    succ m1 zero z -> pred m1 zero z -> col m (A m1 zero z) <> col m z.
```

• e. *Caractérisation* des 0-coutures résiduelles dans `seg1 m` :

```
Theorem succ0_seg1_lb0:forall(m:fmap)(z:dart),
  inv_qhmap m -> let m1 := seg1 m in
    succ m1 zero z <->
     (succ m zero z /\ low_border0 m z (col m (A m zero z))).
```

• f. Définition d'une opération `uA0` renvoyant le premier brin de la 0-orbite de `z` qui est 0-frontière supérieure (éventuellement `nil`) :

```
Lemma not_succ_uA0_nil : forall(m:fmap)(z:dart),
  inv_qhmap m ->
    ~ succ m zero z -> uA0 m z = nil.
Theorem A_seg1_uA0:forall(m:fmap)(z:dart),
  inv_qhmap m -> let m1:=seg1 m in
    succ m1 zero z -> A m1 zero z = uA0 m z.
```

Le théorème affirme la *maximalité* de la première étape de segmentation pour la dimension 0. Idem à la dimension 1.

• g. Question de la *couleur* :

Quasi-hypercartes *bien colorées* :

```
Definition well_coloured(m:fmap):=
  forall(z t:dart),
    expf m z t -> col m z = col m t.
```

Préservation de la bonne coloration par `seg1`:

```
Theorem correct2_seg1:
  forall m:fmap, inv_qhmap m ->
     well_coloured m -> well_coloured (seg1 m).
```

On a donc la *correction totale* de `seg1`.

8.2 Correction de la deuxième étape : seg2

• a. *Terminaison finie* de `seg2_aux` : immédiate.

• b. *Préservation exclusive des brins non isolés*, des *successeurs* et des *prédécesseurs*.

• c. *Bon typage* de `segmentation` : elle préserve à la fois l'invariant de `qhmap` et celui de `hmap`.

• d. *Préservation des couleurs* des brins.

• e. *Préservation de la bonne coloration* d'une hypercarte :

```
Theorem well_coloured_segm:forall(m:fmap),
  inv_hmap m -> well_coloured m->
      well_coloured (segmentation m).
```

• f. *Equivalence* de `up_border` et `low_border` dans une hypercarte :

```
Theorem up_border0_low_border0:forall(m:fmap),
  inv_hmap m -> exd m z
   -> let a:= col m (A m zero z) in
  (up_border0 m z a  <-> low_border0 m z a).

Theorem up_border0_low_border1:forall(m:fmap),
  inv_hmap m -> well_coloured m -> exd m z
   -> let a:= col m (A m zero z) in
  (up_border0 m z a  <-> low_border1 m z a).
```

Alors, les brins de `(segmentation m)` sont ceux qui étaient `low_border0` (ou `up_border0`), ou `low_border1` (ou `up_border1`) dans l'hypercarte `m`.

Ceci achève la preuve de *correction totale* de `segmentation`.

9 Implantation réelle et programme impératif

9.1 Implantation chaînée des hypercartes en C

• Quasi-hypercarte : liste doublement chaînée :

```
#define nil NULL
typedef unsigned char colour;

typedef struct sdart
 {
   struct sdart * s; // list successor
   struct sdart * p; // list predecessor
   struct sdart * A[2]; // 0,1-successors
   struct sdart * A_1[2];// 0,1-predecessors
   colour col;         // colour
 } Sdart, *dart;

typedef dart qhmap, hmap;
```

• Egalité des brins : égalité de pointeurs.

• Prédicats transformés en fonctions booléennes.

• Implantation des *observateurs*, *destructeurs* avec effets de bord, fonctions et prédicats préliminaires : élémentaire.

9.2 Fonction de segmentation en langage C

• Spécifications de `seg1` et `seg2` *indéterministes* et fondées sur un *raisonnement local* concernant les brins et les coutures : *aucun ordre* n'est imposé dans leur parcours.

• *Programmation* de `seg1`, `seg2` ou `segmentation` : simple itération sur la liste des brins avec le traitement local de la spécification.

• *Complexité en temps* $O(n)$, n : nombre de brins. *Complexité en espace* $O(1)$. *Optimalité.*

```
qhmap segmentation(qhmap m)
 {dart x = m,y,x_0,y0,x_1,y1,xs; colour a;
   while(x!=nil)
   {y = A(m,0,x);
    if (y!=nil)
     {m = B(m,0,x);
      if (x!=y)
       {a = col(m,y);
        x_0 = A_1(m,0,x);
        y0 = A(m,0,y);
        if (up_border0(m,y,a))
         {if (low_border0(m,x,a)) m = L(m,0,x,y);
          else m = L(B_1(m,0,x),0,x_0,y);
         }
        else
         {if (low_border0(m,x,a)) m = L(B(m,0,y),0,x,y0);
          else
           {if (x_0==y) m = B(m,0,y);
```

```
            else m = L(B(B_1(m,0,x),0,y),0,x_0,y0);
          }
        }
      }
    }
    // traitement similaire en dimension 1
     xs = (x->s==x||x->s==m)?nil:x->s;
     if (isolated(m,x)) m = D(m,x);
     x = xs;
  }
 return m;
}
```

Très similaire à la spécification.

10 Conclusion

• *Base de spécifications* d'hypercartes pour raisonner avec des subdivisions de surfaces : 10 000 lignes de Coq.

• *Développement pour la segmentation* : environ 5 000 lignes de Coq (100 définitions, théorèmes et lemmes).

• *Programme impératif* écrit "à la main" à prouver. *Extraction de programmes* à partir de preuves.

• *Extensions du problème de segmentation* : dimension 3, surfaces quelconques, en étudiant la variation du genre de l'hypercarte.

• *Algorithmique et modélisation géométriques* : plongements, problèmes numériques.

• *Géométrie discrète* : axiomatique constructive des courbes et surfaces discrètes.