Reasoning Formally with Flip on Plane Triangulations

Jean-François Dufourd[1]

[1]LSIIT, UMR CNRS 7005, Université de Strasbourg, France
(ANR "white" project GALAPAGOS)

Galapagos Days, Sophia-Antipolis, December 10-11, 2009

Introduction

- *Objective*:
  - Building a framework for modeling, reasoning and programming with surface subdivisions (polyhedral surfaces, or polyhedra)
- *Means*:
  - Formal specifications in the Calculus of Inductive Constructions
  - Interactive proofs in the Coq proof assistant (INRIA)
  - A combinatorial hypermap model of polyhedra
- *"Benchmark" of real size*:
  - Triangulations and Flip (Prelimnaries of Delaunay Triangulation)

Outline

Related work

- Planar graphs and triangulations formalized in Isabelle [G. Bauer, T. Nipkow, 2003]
- Combinatorial maps and hypermaps [W.T. Tutte, R. Cori..., 1970-]
- Specification of hypermaps and proof in Coq of the Four Colour Theorem [G. Gonthier et al., 2005]
- Hypermap specification and proof in Coq of Genus Theorem and Euler Formula [J.-F. Dufourd, 2006]
- Design and certification in Coq of an image segmentation algorithm [J.-F. Dufourd, 2006]
- Refinement of topological models in Event-B - Lines and G-maps [C. Dubois, J.-M. Motta, 2007]
- Proof in Coq of a discrete Jordan Curve Theorem (JCT) [J.-F. Dufourd, 2009]
- Formalized proofs of convex hull algorithms [D. Pichardie Y. Bertot, 2001, L.I. Meikle J. Fleuriot, 2004, C. Brun et al., 2009]

Hypermaps in Coq

Definition (*hypermap*)

A *hypermap* is an algebraic structure $M = (D, \alpha_0, \alpha_1)$, where D is a finite set, the elements of which are called *darts*, and α_0, α_1 are permutations on D.

Definition (*Topological cells*)

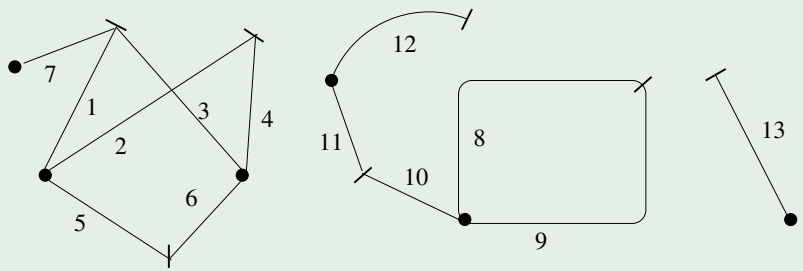
The *topological cells* of a dart x in a hypermap are dart sets which are traversed while iterating some operations from x:
- *edge* of x: iteration of α_0
- *vertex* of x: iteration of α_1
- *face* of x: iteration of $\phi = \alpha_1^{-1} \circ \alpha_0^{-1}$
- *connected component* of x: iteration of both α_0 and α_1.

(*Projection, embedding*)

- In a *projection* of a hypermap onto a surface: vertices and edges are projected onto points, darts onto open Jordan curves, faces onto open connected regions.
- An *embedding* is a projection without self-intersection which determines a *subdivision* of the surface.

Example: A hypermap projected onto a plane (without self-intersections)

D	1	2	3	4	5	6	7	8	9	10	11	12	13
α0	3	4	7	2	6	5	1	9	8	11	10	12	13
α1	5	1	6	3	2	4	7	10	8	9	12	11	13



13 darts, 7 edges (*strokes*), 6 vertices (*bullets*), 4 faces, and 3 connected components.

Inductive definition of a type `dim` of *dimensions*

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

Definition of a type `dart` of *darts* as a renaming of `nat`

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

Definition of a type `point` of the Euclidean plane

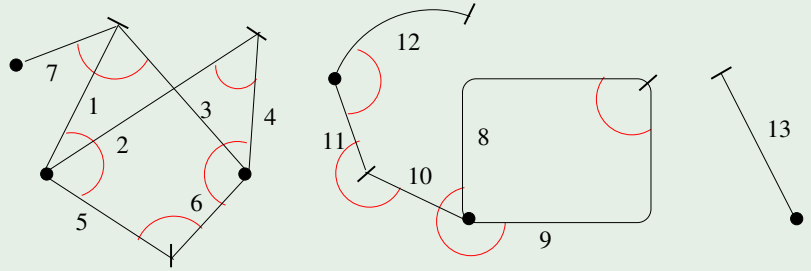
```
Definition point:= R * R.
Definition origin:= (0,0).
```

Inductive definition of a type `fmap` of *free maps* (free algebra of terms)

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

Example: Description of the previous hypermap. Links are represented by *arcs of circle*. The orbits are (intentionally) *incompletely linked*

D	1	2	3	4	5	6	7	8	9	10	11	12	13
α0	3	4	7	2	6	5	1	9	8	11	10	12	13
α1	5	1	6	3	2	4	7	10	8	9	12	11	13



Observers of free maps

- *Existence* `(exd m z)` *of a dart* `z` in a free map `m`:

```
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.
```

- *Operation* α_k: `(cA m k z)` returns the *k-successor* of `z` in `m`.
- *Existence* `(succ m k z)` *of a k-successor* of `z` inserted by `L`.

Example: successors in the hypermap example

```
cA m one 1 = 5, cA_1 m one 5 = 1,
succ m zero 2,pred m one 2,~succ m one 1
```

Destructors in a free map

- *Deletion* `D` of a dart `z` in `m`: `(D m z)`
- *Break* `B` in `m` of a link starting from `z` at dimension `k` *(deletion of an arc of circle)*: `(B m k z)`

Inverses or *symmetrical operations*

`A_1`, `pred`, `cA_1`, `B_1`, `F_1`, `cF_1`...

Hypermaps as a subtype of the free maps

- *Precondition* on `I`: `(prec_I m x)` expresses that `x` is different from `nil` and does not exist in `m`.
- *Precondition* on `L`: `(prec_L m k x y)` expresses that `x` and `y` both exist in `m`, `x` has no `k`-successor, `y` has no `k`-predecessor, and that their `k`-orbit will stay open.
- *Invariant* of the *hypermaps* (with *open* edges and vertices):

```
Fixpoint inv_hmap(m:fmap):Prop:=
 match m with
    V => True
  | I m0 x _ => inv_hmap m0 /\ prec_I m0 x
  | L m0 k0 x y => inv_hmap m0 /\ prec_L m0 k0
 end.
```

- *Properties*: for any `m` and `k`, `cA` is a *permutation* and `cA_1` is its *inverse*.

Orbits, cells and planarity

Let M be a hypermap, and d, e, v, f, c be its *numbers* of *darts, edges, vertices, faces, connected components*, respectively.

Definition (*Euler characteristic, genus, planarity, Euler formula*)

(i) The *Euler characteristic* of M is $\chi = v + e + f - d$.
(ii) The *genus* of M is $g = c - \chi/2$.
(iii) When $g = 0$, the hypermap is said to be *planar*.
(iv) A planar hypermap satisfies the *Euler formula*:

$$\chi = v + e + f - d = 2 * c$$

.

Example

For the example hypermap, $\chi = 6 + 7 + 4 - 13 = 4$ and $g = 3 - \chi/2 = 1$. Consequently, the hypermap is non-planar.

Theorem *of the genus*

(i) χ is an even integer.
(ii) g is a non-negative integer.

Interpretation of the genus

The *genus* corresponds with the minimal *number of holes* in an orientable closed surface the hypermap can be embedded onto (without self-intersection).

Example

The example hypermap with genus 0 cannot be embedded onto a *plane*. It can be embedded on a *torus* (of genus 1).

Specifications of orbits

- They are genericly defined by Coq *signatures* and *modules* for any inverse bijections `f` and `f_1` in a hypermap.
- A *specialization* is done for `(cA m zero)`, `(cA m one)`, `(cF m)` and their inverses.
- The *existence of a path* in an orbit from dart `x` to dart `y` is easy to define, e.g. *in an edge, a vertex, a face*: `expe m x y`, `expv m x y`, `expf m x y`

Connectivity

- The membership of `x` and `y` to the *same connected component* is expressed by: `eqc m x y`

Proven properties

- Each orbit is *periodic* with a *uniform lowest period*.
- `(expe m)`, `(expv m)`, `(expf m)` and `(eqc m)` are *decidable equivalences*.

Degrees

- The number of darts of an orbit of `x` in `m` is its *degree*: e.g. *for an edge, a vertex, a face*: `degree m x`, `degrev m x`, `degref m x`
- `The degrees is the same for each dart of an orbit.`

Numbers of darts, edges, vertices, faces, components

```
Fixpoint nd(m:fmap):Z:=
 match m with
    V => 0
  | I m0 x _ => nd m0 + 1
  | L m0 _ _ _ => nd m0
 end.

Fixpoint ne(m:fmap):Z:=
 match m with
    V => 0
  | I m0 x _ => ne m0 + 1
  | L m0 zero x y => ne m0 -
   if eq_dart_dec (cA m0 zero x) y then 0 else 1
  | L m0 one x y => ne m0
 end.

 (* Idem for nv, nf, nc *)
```

Euler characteristic, genus, planarity

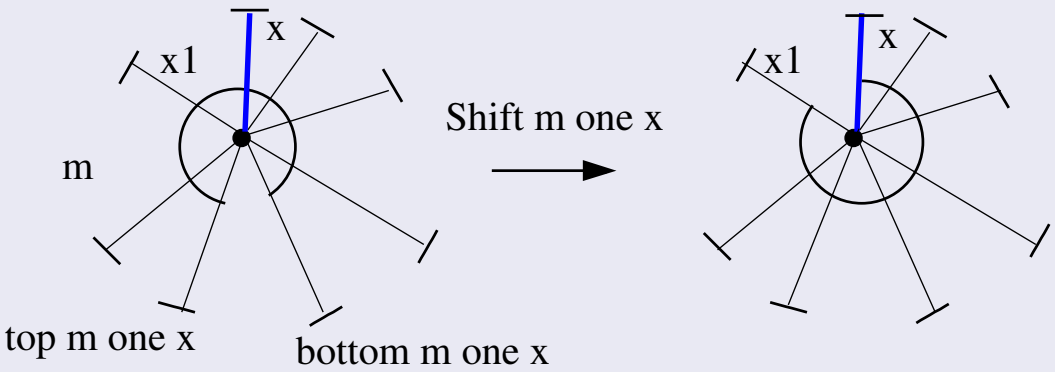
```
Definition ec(m:fmap): Z:=
   nv m + ne m + nf m - nd m.

Definition genus(m:fmap): Z:=
  (nc m) - (ec m)/2.

Definition planar(m:fmap): Prop:=
   genus m = 0.
```

Genus Theorem and Euler Formula

```
Theorem Genus_Theorem: forall m:fmap,
 inv_hmap m -> genus m >= 0.
Proof. (* by induction on m *).

Lemma Euler_Formula: forall m:fmap,
 inv_hmap m -> (planar m <-> ec m / 2 = nc m).
Proof. (* trivial *).
```

Split

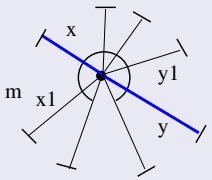
Shifting an orbit opening



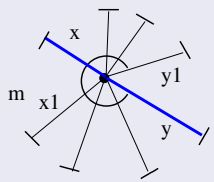

```
Definition Shift(m:fmap)(k:dim)(x:dart):=
  L (B m k x) k (top m k x) (bottom m k x).
```

Splitting a k-orbit

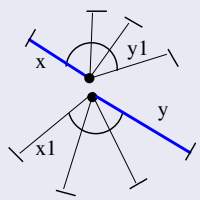
```
Definition crackv(m:fmap)(x y:dart):Prop:=
   x <> y /\ expv m x y.
```



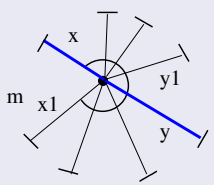
Case 1: succ m one x ∧ succ m one y



Case 2: succ m one x ∧ ~ succ m one y



Split m one x y



Case 3: ~ succ m one x ∧ succ m one y

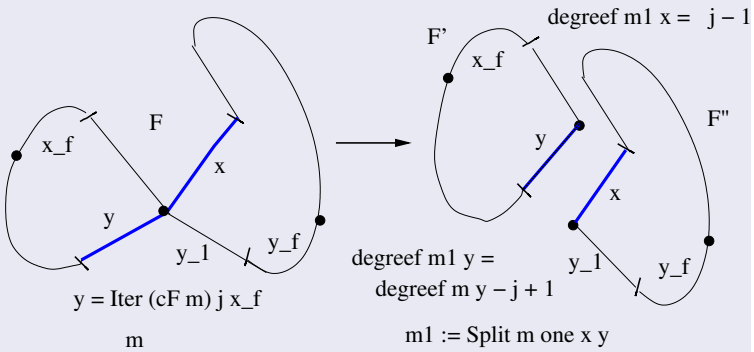
```
Definition Split(m:fmap)(k:dim)(x y:dart):fmap:=
  if succ_dec m k x
  then if succ_dec m k y
       then B (Shift m k x) k y       (* 1 *)
       else B m k x                   (* 2 *)
  else B m k y.                       (* 3 *)
```

Evolution of the number of orbits

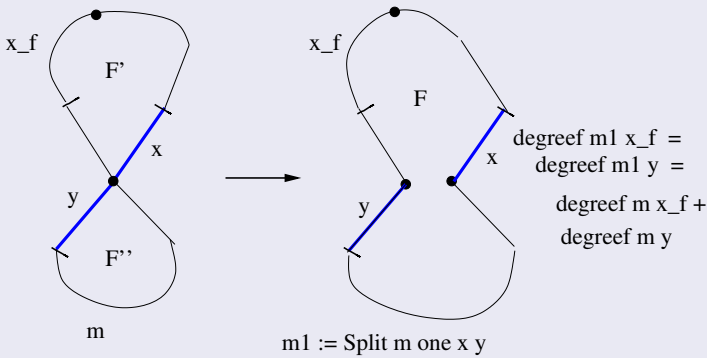
For example, number of faces:

```
Theorem nf_Split1: forall (m:fmap)(x y:dart),
  inv_hmap m -> crackv m x y ->
    nf (Split m one x y) = nf m +
      if expf_dec m x y then 1 else -1.
```

Evolution of the faces during a vertex splitting



Case 1 (x and y in the same face): one more face.



Case 2 (x and y are in two distinct faces): one face removed.

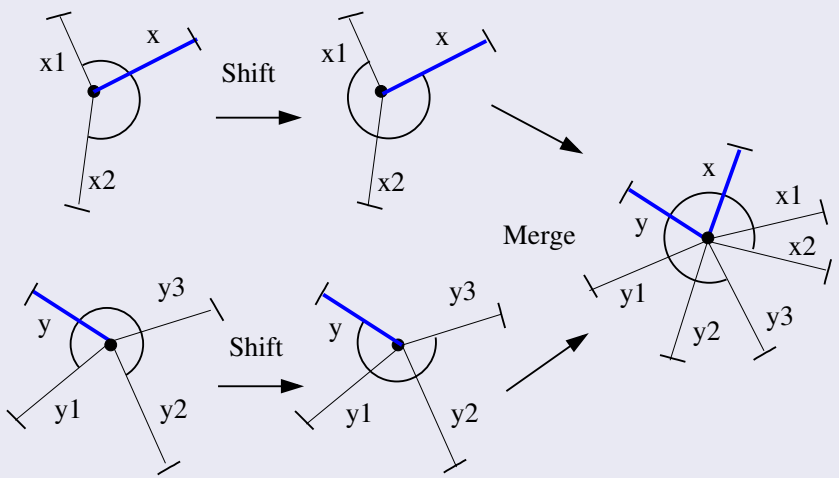
Criteria of planarity and connectivity

```
Theorem planarity_crit_Split1:
 forall(m:fmap)(x y:dart),
  inv_hmap m -> crackv m x y ->
    let m0 := Split m one x y in
(planar m <->
  (planar m0 /\ (~eqc m0 x y \/ expf m0 x y))).

Theorem disconnect_planar_criterion_Split1:
 forall (m:fmap)(x y:dart),
  inv_hmap m -> planar m -> crackv m x y ->
(expf m x y <->
    ~eqc (Split m one x y) x (cA m one x)).
```

Merge

Glueing and Merging two k-orbits

```
Definition G(m:fmap)(k:dim)(x y:dart):=
  let m0:= if succ_dec m k x
           then Shift m k x else m in
  L m0 k x y.

Definition Merge(m:fmap)(k:dim)(x y:dart):=
  let m1 := if pred_dec m k y
            then Shift m k (cA_1 m k y)
            else m in
  G m1 k x y.
```

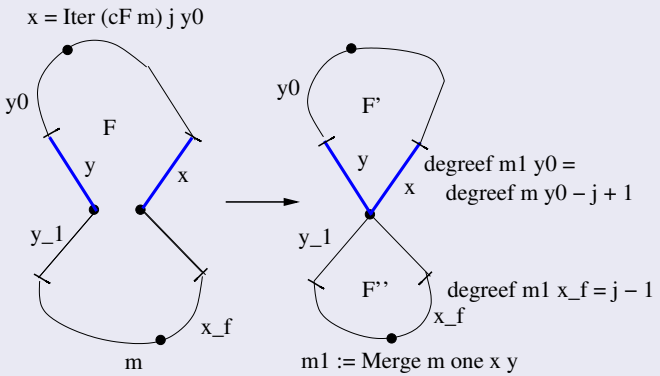


Evolution of the number of orbits

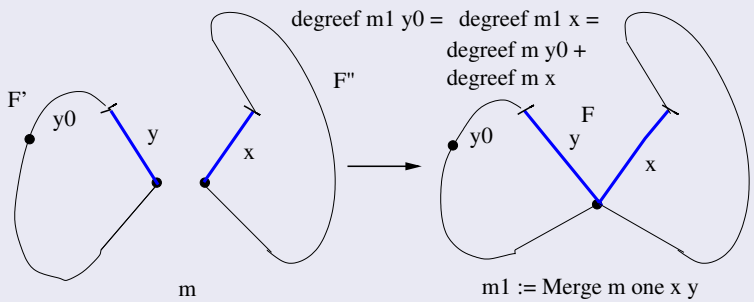
For example, number of faces:

```
Theorem nf_Merge1:
forall (m : fmap) (x y: dart) (H : inv_hmap m),
 exd m x -> exd m y ->
  let y0 := cA m zero y in
    nf (Merge m one x y) = nf m +
      if (expf_dec m x y0) then 1 else -1.
```

Evolution of the faces during a merging of vertices



Case 1 (y0 and x in the same face): one more face.



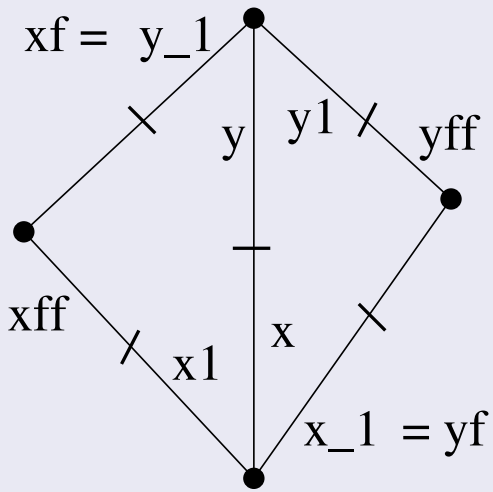
Case 2 (y0 and x in two distinct faces): one face removed.

Merging and planarity

```
Theorem genus_variation_Merge1:
forall(m:fmap)(x y:dart),
  inv_hmap m -> exd m x -> exd m y ->
   ~expv m x y ->
     genus (Merge m one x y) = genus m +
      if eqc_dec m x y
      then if expf_dec m x (cA m zero y)
           then 0
           else 1
      else 0.

Theorem planarity_crit_Merge1:
forall (m : fmap) (x y : dart),
 inv_hmap m -> exd m x -> exd m y ->
  ~ expv m x y ->
(planar (Merge m one x y) <->
 planar m /\
   (~ eqc m x y \/ expf m x (cA m zero y))).
```

Flip of an edge

Flip configuration

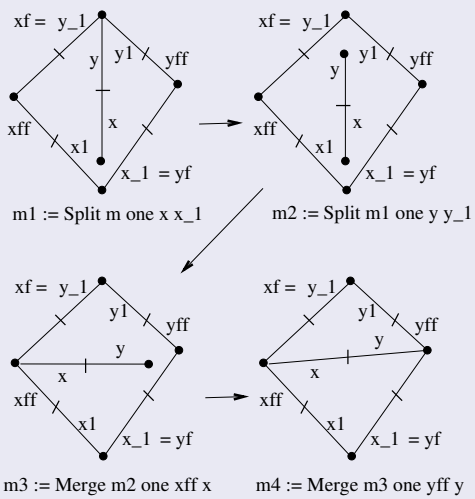



```
Definition prec_Flip(m:fmap)(x:dart):Prop:=
  let y:= cA m zero x in
   exd m x /\ ~ expf m x y /\ ~ expv m x y /\
     3 <= degreev m x /\ 3 <= degreev m y.
```

Flip definition

```
Definition Flip(m:fmap)(x:dart):fmap:=
  let x_1 := cA_1 m one x in
  let y := cA m zero x in
  let y_1 := cA_1 m one y in
  let xf := cF m x in
  let xff := cF m xf in
  let yf := cF m y in
  let yff := cF m yf in
  let pxff := fpoint m xff in
  let pyff := fpoint m yff in
  let m1 := Split m one x x_1 in
  let m2 := Split m1 one y y_1 in
  let m3 := Merge m2 one xff x in
  let m4 := Merge m3 one yff y in
  let m5 := Chp m4 x pxff in
  let m6 := Chp m5 y pyff in
in m6.
```

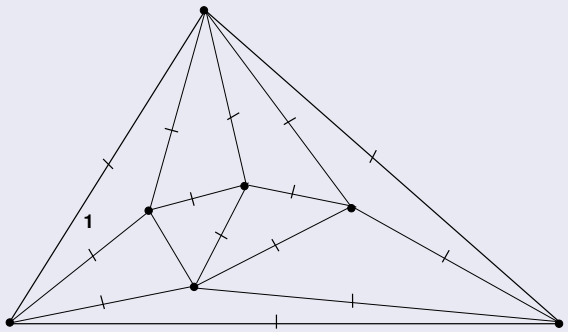
Flip illustration



Flipping in a topological triangulation

Definition (*Topological triangulation*)

A *topological triangulation* is a hypermap with the following properties:
(1) each *edge* contains exactly two darts;
(2) each *vertex* contains at least two darts;
(3) each *face* contains exactly three darts, i.e. is a *triangle*, maybe except for one of them which must have *at least* three darts.



Formalization

```
Definition isMap(m:fmap):Prop:=
  forall z:dart, exd m z -> degreee m z = 2.
Definition isPoly(m:fmap):Prop:=
  forall z:dart, exd m z -> 2 <= degreev m z.
Definition isTri(m:fmap)(z:dart):Prop:=
  degreef m z = 3.
Definition isTriangulation(m:fmap):Prop:=
 forall z:dart, exd m z -> isTri m z.
Definition inv_Triangulation(m:fmap):Prop:=
  inv_hmap m /\ isMap m /\ isPoly m /\
    isTriangulation m.
```

Topological properties

```
Definition isQuad(m:fmap)(z:dart):Prop:=
  degreef m z = 4.
Definition isBar(m:fmap)(z:dart):Prop:=
  degreee m z = 2 /\ degreev m z = 1 /\
    degreef m z = 2.
Definition isHexa(m:fmap)(z:dart):Prop:=
  degreef m z = 6.
```

Topological properties: invariant, planarity

```
Theorem inv_Triangulation_Flip:
 forall(m:fmap)(x:dart),
   inv_Triangulation m -> prec_Flip m x ->
      inv_Triangulation (Flip m x).

Theorem planar_Flip:
  forall(m:fmap)(x:dart),
   inv_Triangulation m -> prec_Flip m x ->
     planar m -> planar (Flip m x).
```

Some details

- *Successors in faces:*

```
cF m1 y = x /\ cF m1 x = y_1 /\
 cF m1 y_1 = xff /\ cF m1 xff = x_1 /\
   cF m1 x_1 = yff /\ cF m1 yff = y.
```

- *Other successors in faces:*

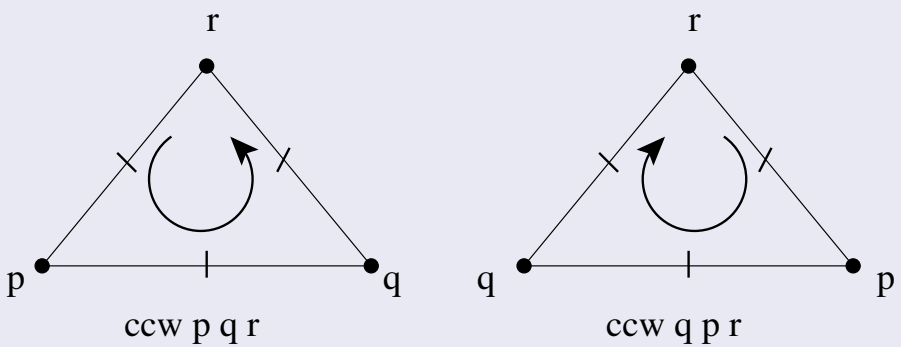
```
... forall z, exd m z ->
   x <> z -> y <> z -> y_1 <> z ->
     xff <> z -> x_1 <> z -> yff <> z ->
       cF m1 z = cF m z.
```

- *Other faces (cont'd):*

```
... forall z, exd m z -> .
     ~expf m1 x z -> isTri m1 z
```

Flipping in a plane embedded triangulation

Triangle orientation: ccw



a. Counterclockwise orientation of (p,q,r).

b. Clockwise orientation of (p,q,r).

Knuth axiomatics and Euclidean model.

Translation in hypermaps

```
Definition ccw_triangle(m:fmap)(z:dart):Prop:=
  let pz:= fpoint m z in
  let pzf:= fpoint m (cF m z) in
  let pzff:= fpoint m (cF m (cF m z)) in
      ccw pz pzf pzff.
```

Definition (*Well-embedded triangulation*)

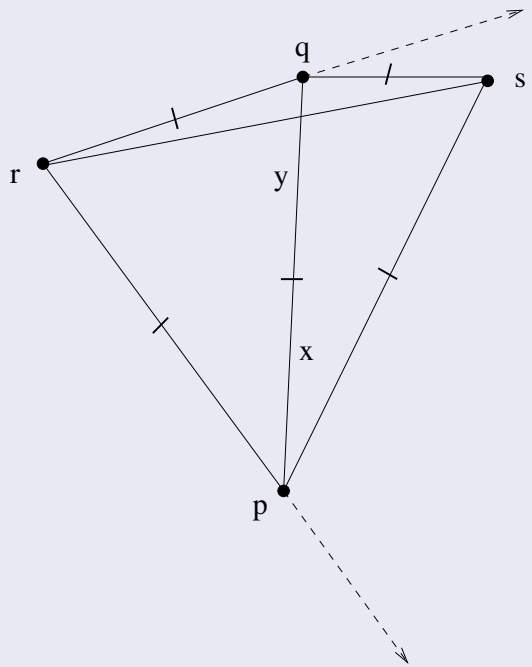
A topological triangulation is *well-embedded on the plane* if:
(1) the (two) darts of each *edge* are embedded on distinct points;
(2) the darts of each *vertex* are embedded on the same point;
(3) the (three) darts of each *face* are embedded as a counterclockwise oriented triangle, except for the face of 1 which is embedded as a clockwise oriented triangle.

Translation in Coq's hypermap

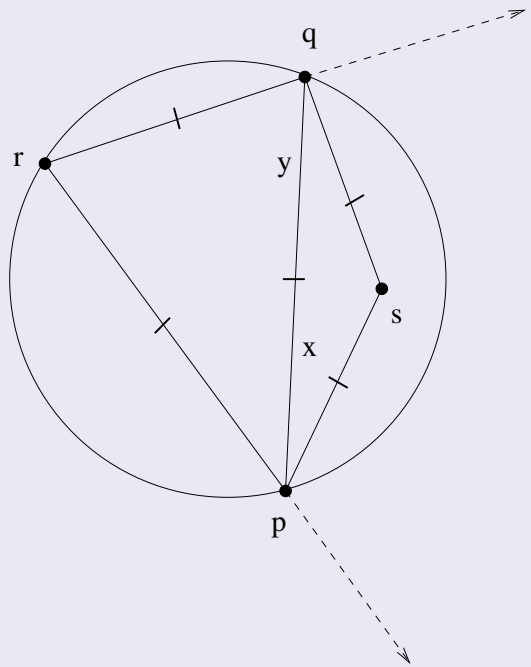
```
Definition prec_Flip_emb(m:fmap)(x:dart):Prop:=
 let y := cA m zero x in
 let px := fpoint m x in
 let py := fpoint m y in
 let pxff := fpoint m (cF m (cF m x)) in
 let pyff := fpoint m (cF m (cF m y)) in
    ccw px py pxff /\ ccw py px pyff /\
       ccw px pyff pxff /\ ccw py pxff pyff.

Definition isWellembede(m:fmap):Prop:=
  forall z:dart, exd m z ->
    fpoint m z <> fpoint m (cA m zero z).
Definition isWellembedv(m:fmap):Prop:=
  forall z t:dart, exd m z -> expv m z t ->
    fpoint m z = fpoint m t.
Definition isWellembedf(m:fmap):Prop:=
  cw_triangle m 1 /\
    forall z, exd m z -> ~expf m z 1 ->
      ccw_triangle m z.
Definition isWellembed(m:fmap):Prop:=
 exd m 1 /\ isWellembede m /\
   isWellembedv m /\ isWellembedf m.
```

Definition (*Illegal edge in a triangulation*)

An edge is *illegal* in a well-embedded plane triangulation when:
(1) its two adjacent triangles are counterclockwise oriented (which excludes the external face);
(2) the vertex of one of the two triangles which is not extremity of the common edge is strictly in the circumcircle of the other triangle.



a. Angular sector (rp, rq) allowed for s.



b. s is in the circumcircle of (p,q,r).

Preservation of a well-embedded triangulation

```
  Theorem prec_Flip_emb_prec_Flip:
   forall (m:fmap)(x:dart),
    inv_Triangulation m -> isWellembede m ->
     isWellembedv m -> exd m x ->
       prec_Flip_emb m x -> prec_Flip m x.

. Theorem isWellembed_Flip:
   forall (m:fmap)(x:dart),
    inv_Triangulation m -> isWellembed m ->
      exd m x -> prec_Flip_emb m x ->
       isWellembed (Flip m x).
```

Conclusions

We have:

- enriched by Split, Merge and Flip our framework of *hypermap specifications*
- characterized *planar triangulations* and *Flip* from topological and geometrical points of view
- proved the properties of Flip

The development counts $65,000$ Coq lines, with more than 250 definitions and 600 lemmas and theorems.

Future work

The next step will be:

- to formally characterize *Delaunay triangulations* with hypermaps
- to write in Coq *algorithms* to build them and to prove their *correctness*.

Recent references

- Jean-François Dufourd: An Intuitionistic Proof of a Discrete Form of the Jordan Curve Theorem Formalized in Coq with Combinatorial Hypermaps. *Journal of Automated Reasoning* **43(1)**: 19-51 (2009).
- Jean-François Dufourd: Polyhedra genus theorem and Euler formula: A hypermap-formalized intuitionistic proof. *Theoretical Computer Science* **403(2-3)**: 133-159 (2008).
- Jean-François Dufourd: Design and formal proof of a new optimal image segmentation program with hypermaps. *Pattern Recognition* **40(11)**: 2974-2993 (2007).