Hypermap Specification and Linked Implementation Certification using Orbits

Jean-François Dufourd

ICUBE Laboratory, University of Strasbourg - CNRS, France

FLOC - ITP’2014, Vienna
Introduction

Aim:
- Specify libraries and certify pointer implementations

Use of:
- Algebraic datatypes
- Calculus of Inductive Constructions (CIC) and Coq system
- Simulation of a fragment of C
- Orbits [Dufourd 14, soon in TCS...]:
  central in specification and implementation

Non-use of:
- Floyd-Hoare logic (and Separation logic)

Case study, focused on orbits:
- Combinatorial hypermaps [Cori 70, ..., Gonthier 08, ...]

Applications in Combinatorics and Geometric modeling

Dedicated to George Gonthier...
Orbits in Coq

Context
- A type \( X \), with a decidable equality \( \text{eqd} \ \ X \rightarrow X \rightarrow \text{Prop} \)
- A total function \( f : X \rightarrow X \)
- A finite subdomain \( D : \text{list} \ X \)
  (represented as a finite list without duplication)

Definitions, Notations
For any \( z : X \), consider the \( f \)-iterates:
\[
z_0 := z, \ z_1 := f \ z, \ldots, \ z_k := f \ z(k-1), \ldots
\]
It is proved that there is a smallest \( p \) such that \( z_p \) is not in \( D \) or \( z_p \) is already met among \( z_0, \ldots, \ z(p-1) \).
(i) \( f \)-orbit of \( z \): \( \text{orb} \ \ X \ f \ D \ z := [z_0 \ldots z(p-1)] \) if \( z \) is in \( D \), [ ] otherwise
(ii) length of \( z \)'s \( f \)-orbit: \( \text{lorb} \ \ X \ f \ D \ z := p \)
(iii) \( f \)-limit of \( z \): \( \text{lim} \ \ X \ f \ D \ z := z_p \)
Orbits

**Orbit shapes**

The *orbit* of $z : X$ can be:

(i) **empty**: $\sim \text{In } z \ D$

(ii) **line**: $\text{inv\_line } X \ f \ D \ z$

(iii) **a (closed) crosier**: $\text{inv\_crosier } X \ f \ D \ z$

(iv) **a circuit**: $\text{inv\_circ } X \ f \ D \ z$

**Example: Orbits (4 positions of $z$)**

![Example diagram showing orbit shapes](image-url)
Connected component shapes

The orbits of all $z$ of $D$ are separated or in collision. In fact, $(D, f)$ is a functional graph, each connected component being:

(i) either a tree

(ii) or a circuit on which trees are grafted

Example: Components (2 positions of $z$)
**Definitions: Inverse, closure**

- \( f \_1 \ X \ f \ D \ z : \text{inverse} \) of \( f \) at \( z \), when \( z \) has only one \( f \)-predecessor in \( D \)
- \( \text{Cl} \ X \ f \ D : \text{closure} \) of \( f \), when all components are (linear) branches or circuits (\( f \) : partial injection in \( D \))

**Example: Inversion, closure**

![Diagram](image)

**Figure**: (a) Branch containing \( z \) / (b) Inversion / (c) Closure.
**Addition/deletion**

*Addition* of a new element \( a \) in \( D \), when \( \sim \) In a \( D \).

*Deletion* of an element \( a \) from \( D \).

**Example: Addition/deletion**

*Figure*: Addition (Left) / Deletion (Right).
Mutation

A \textit{mutation} modifies the \textit{f-image} $f(u)$ of an element $u$ into an element $u_1$ while all the other images do not change:

\begin{verbatim}
Definition Mu(f:X->X)(u u1:X)(z:X):X :=
  if eqd X u z then u1 else f z.
\end{verbatim}

Example: Mutation

\begin{figure}
\centering
\begin{tikzpicture}
  % Diagram code here
\end{tikzpicture}
\caption{Mutation: Cases A and B.}
\end{figure}
**Transposition**

A *transposition* exchanges the $f$-images of two elements in circuits and do not change the others (only one element $u$ and its new image $u_1$ are precised):

\[
\text{Definition } Tu(f:X \rightarrow X)(D: \text{list } X)(u \ u_1:X)(z:X):X := \\
\quad \text{if eqd } X \ u \ z \ \text{then } u_1 \\
\quad \text{else if eqd } X \ (f\ _1 X \ f \ D \ u_1) \ z \ \text{then } f \ u \ \text{else } f \ z.
\]

When:
- $u$ and $u_1$ are in the same circuit: *split* into two circuits
- $u$ et $u_1$ are in two circuits: *merge* into one circuit

**Example: Transposition**

![Diagram showing transposition](image)

**Figure**: Split (Left) / Merge (Right).
Hypermaps in mathematics

**Definition**

A **combinatorial (2-dimensional) hypermap** is an algebraic structure, $M = (D, \alpha_0, \alpha_1)$, where:

- $D$ is a finite set, the elements are called *darts*,
- and $\alpha_0, \alpha_1$ are two *permutations* on $D$ indexed by a *dimension*, 0 or 1.

**Example: hypermap**

<table>
<thead>
<tr>
<th>D</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_0$</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>9</td>
<td>12</td>
<td>11</td>
<td>14</td>
<td>13</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>$\alpha_1$</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>8</td>
<td>4</td>
<td>16</td>
<td>11</td>
<td>10</td>
<td>13</td>
<td>12</td>
<td>15</td>
<td>14</td>
<td>9</td>
</tr>
</tbody>
</table>

**Figure**: Hypermap embedded in the plane (dart embedding: a Jordan arc beginning by a bullet, ending by a small strike). 

J.-F. Dufourd (Strasbourg, ICUBE)
Mathematical recalls

**Orbits/components of hypermaps**

The *edge* (resp. *vertex*, *face*, *component*) of $z$ is its connected component in the graph $(D, \alpha_0)$ (resp. $(D, \alpha_1)$, $(D, \alpha_1^{-1} \circ \alpha_0^{-1})$, $(D, \{\alpha_0 \cup \alpha_1\})$)

(roughly: edge = small strike, vertex = bullet...).

**Classification**

The hypermaps are *classified* according to their numbers of edges, vertices, faces and components, thanks to the notions of *Euler characteristic*, *genus* and *planarity*. 
Darts, dimensions, hypermaps

(* Type of darts and exception: *)
Definition dart := nat.
Definition nild := 0.
...

(* Type of dimensions: *)
Inductive dim:Type :=
  zero : dim | one : dim.

(* Type of hypermaps: *)
Inductive hmap:Type :=
  V : hmap (* Void (empty) hmap *)
  | I : hmap->dart->hmap (* Insertion of a dart *)
  | L : hmap->dim->dart->dart->hmap. (* k-Linking from a dart to another *)

Example: Hypermap

m1 := I ( I ( I ( I ( I V 1) 2) 3) 4) 5) 6.
m2 := L (L m1 zero 4 2) zero 2 5).
m3 := L (L (L m2 one 1 2) one 2 3) one 6 5.

Figure: Partial coding of the example hypermap (The 0- and 1-orbits stay "open" in the specification).
**Free map observers (inductively defined)**

- **Existence** of a dart \( z \) in the hmap \( m \): \( \text{exd} \ m \ z \)
- **\( k \)-successor** of \( z \) in \( m \): \( \text{pA} \ m \ k \ z \)

returns \( \text{nild} \) when there is no \( k \)-link from \( z \)

- **\( k \)-successor** of \( z \) **in the closure** of \( \text{pA} \ m \ k \ A \ m \ k \ z \)

Note: two successor notions which are useful in the specification

**Example: Zoom on an edge of hypermap**

![Diagram](image_url)

**Figure**: An edge: open for \( \text{pA} \), closed for \( A \).

**Preconditions, invariant of hypermaps**

- **Preconditions** on \( I \) and \( L \) impose that edges and vertices remain open for \( \text{pA} \)
- An **invariant** of hypermaps \( \text{inv\_hmap} \ m \) derives.
Orbital properties of hypermaps

**Idea:** studying $pA$ et $A$ by the properties of their *orbits*:
- The $pA$-orbits stay (open) *lines*
- The components w.r.t. $pA$ are *branches*
- $A$ is really the *closure* of $pA$
- The $A$-orbits are (closed) *circuits*
- The same for the inverses $pA^{-1}$ and $A^{-1}$

Other properties of the hypermaps

All this leads to fundamental results on discrete topology:
- Incremental definitions of *numbers of edges, vertices, faces, components, Euler characteristic, genus* and *planarity*
- An inductive proof of the *Genus theorem*
- Constructive criteria of *planarity*
- A proof of the *discrete Jordan curve theorem*
**Addresses**

- The potential *addresses* are the natural numbers
- There is an *exception address* null (= 0)

(* Address type: *)

Definition Addr := nat.

(* Exception: *)

Definition null := 0.

**Memories/Validity**

- A memory is *non-bounded* and the allocations always succeed
- It is *partitioned* according to the *datatypes*

(* Contexte: *)

Variables (T:Type) (undef:T).

(* Memory type: *)

Inductive Mem:Type:=
- initm : Mem ( * empty memory *)
- insm : Mem->Addr->T->Mem. ( * insertion of (address, value) * )

(* Validity domain: *)

Fixpoint dom(M:Mem)(z:Addr):list Addr := ...

(* From where a precondition on insm and an invariant inv_Mem... *)
**Formalization of Memory**

**Formalization of Memory (II)**

### Address generation

- A *fresh address* (invalid and non-null) can always be generated by a function we call *adgen*:

  Parameter adgen: Mem->Addr.

### Conservative memory operations

- **allocation**: `alloc M` returns M updated and a fresh address:

  Definition `alloc(M:Mem):(Mem * Addr)%type :=
  let a := adgen M in (insm M a undef, a).

  Inductively specified:
  - **loading**: `load M z`
  - **mutation**: `mut M z t`
  - **releasing**: `free M z`
Cells for the darts, memories of dart cells

(* Dart cell type: *)

Record cell:Type:=
    mkcell { s : dim->Addr; (* "array" of 2 k-successors *)
            p : dim->Addr; (* "array" of 2 k-predecessors *)
            next : Addr (* successor *)
        }.

(* Type of cell memories: *)

Definition Memc := Mem cell.

Example: A dart cell

Figure: Cell of a hypermap representation in a memory M.
Hypermap Representation (II)

Hypermap representation

Any hypermap representation \( R_m \) is a pair composed of:

- a **cell memory** \( M : \text{Memc} \)
- a **pointer** (head of a main list) \( h : \text{Addr} \)

(* Type of hypermap representations: *)

\[
\text{Definition } \text{Rhmap} := (\text{Memc} \times \text{Addr})\%\text{type}.
\]

Observers

(on \( \text{Rhmap} \), names are prefixed by "R";
on \( \text{Memc} \), names are suffixed by "c")

(* Observers of the main list: *)

\[
\text{Definition } \text{Rnext } M z := \text{next} (\text{loadc } M z).
\]

\[
\text{Definition } \text{Rorb } Rm := \text{let } (M, h) := Rm \text{ in}
\]

\[
\quad \text{orb } \text{Addr} (\text{Rnext } M) (\text{domc } M) h.
\]

\[
\text{Definition } \text{Rlim } Rm := \text{let } (M, h) := Rm \text{ in}
\]

\[
\quad \text{lim } \text{Addr} (\text{Rnext } M) (\text{domc } M) h.
\]

(* Observers of hypermap: *)

\[
\text{Definition } \text{Rexd } Rm z := \text{In } z (\text{Rorb } Rm).
\]

\[
\text{Definition } \text{RA } M k z := s (\text{loadc } M z) k.
\]

\[
\text{Definition } \text{RA}_1 M k z := p (\text{loadc } M z) k.
\]

...
Invariant of representation

For any hypermap representation $R_m$, some features are required:

1. A *main singly-linked list* of darts representations: a *line* with valid darts and a null *limit*:

   Definition $\text{inv\_Rhmap1}(R_m:\text{Rhmap}) := \text{let } (M, h) := R_m \text{ in}$
   
   $\text{inv\_Memc} M \underset{\text{\lor}}{\text{\lor}} (h = \text{null} \underset{\text{\lor}}{\text{\lor}} \text{In} h (\text{domc} M)) \underset{\text{\lor}}{\text{\lor}} \text{Rlim} R_m = \text{null}.$

2. For each dart, 4 *circular singly-linked lists* for the $k$-links: each one is a *circuit* with darts in the main list, and $\text{RA}_1 M k$ is always the *inverse* of $\text{RA} M k$:

   Definition $\text{inv\_Rhmap2}(R_m:\text{Rhmap}) := \text{let } (M, h) := R_m \text{ in}$
   
   $\forall k \forall z, \text{Rxd} R_m z \rightarrow$
   
   $\text{inv\_circ} \text{Addr} (\text{RA} M k) (\text{Rorb} R_m) z \underset{\text{\lor}}{\text{\lor}}$
   
   $\text{RA}_1 M k z = f\_1 \text{Addr} (\text{RA} M k) (\text{Rorb} R_m) z.$

   Definition $\text{inv\_Rhmap}(R_m:\text{Rhmap}) := \text{inv\_Rhmap1} R_m \underset{\text{\lor}}{\text{\lor}} \text{inv\_Rhmap2} R_m.$
User update operations (I): Empty hypermap

_Ideas:_
- providing a set of specified operations which must be: _conservative_ (w.r.t. invariants), _minimal, complete, ready-to-assemble, safe_ (hiding pointer manipulations).
- simulating the C language

**Empty hypermap: RV**

Definition RV(M:Memc): Rhmap := (M, null).

**Properties**

Correct behavior w.r.t. the observer _Rexd_:

Lemma Rexd_RV: forall M z, inv_Memc M -> ~Rexd (RV M) z.
**Insertion of a new isolated dart: $RI$**

Definition $RI(Rm:Rhrefmap):Rhrefmap :=$

\[
\text{let } (M, h) := Rm \text{ in } \\
\text{let } (M1, x) := \text{alloc} M \text{ in } \\
\text{let } M2 := \text{mut} M1 \ x \ (\text{modnext} (\text{ficell} \ x) \ h) \text{ in } (M2, x).
\]

**Example: Insertion of an isolated dart**

![Diagram](image)

**Figure**: $RI$: insertion of a dart in $Rm$ (Left) giving $RI \ Rm$ (Right).

**Properties**

- Correct observational behavior w.r.t. $Rexd, RA, RA_1$, e.g.:
  
  For any $k$, the new dart is a fixpoint w.r.t. $RA \ M2 \ k$ and $RA_1 \ M2 \ k$

- Proofs inherited from the general orbits: addition and mutation properties
User update operations (III): Transposition

**Transposition of two darts at dimension k: RL**

Definition \( RL(Rm:Rhmap)(k:dim)(x y:Addr): Rhmap := \)
\[
\text{let } (M, h) := Rm \text{ in } \\
\text{let } x_k := RA M k x \text{ in let } y_k := RA_1 M k y \text{ in } \\
\text{let } M3 := mutc M x (mods (loadc M x) k y) \text{ in } \\
\text{let } M4 := mutc M3 y (modp (loadc M3 y) k x) \text{ in } \\
\text{let } M5 := mutc M4 y_k (mods (loadc M4 y_k) k x_k) \text{ in } \\
\text{let } M6 := mutc M5 x_k (modp (loadc M5 x_k) k y_k) \text{ in } \\
(M6, h).
\]

Definition \( \text{prec}_R L Rm k x y := \text{In } x (Rorb Rm) \land \text{In } y (Rorb Rm). \)

**Exemple: Transposition of two darts \( x \) and \( y \) at dimension 1**

**Figure**: RL: Transposition in \( Rm \) (Left) giving \( RL Rm \ one \ x \ y \) (Right).

**Properties**

- **Correct observational behavior w.r.t.** \( R\text{exd}, RA, RA_1 \), e.g.:
  - \( y \) is the new \( k \)-successor of \( x \): RL splits or merges (circular) \( k \)-orbits w.r.t. \( RA \ M \ k \) and \( RA_1 \ M \ k \)
- **Proofs inherited from the general orbits: transposition properties**
**Deletion of an isolated dart: RD**

Definition \( RD(Rm:Rhmap)(x:Addr)(H: \text{inv}_Rhmap1 \ Rm): Rhmap := \)

let \((M,h) := Rm\) in

if \( \text{eqd} \ Addr \ h \ \text{null} \) then \( Rm \)

else if \( \text{eqd} \ Addr \ x \ \text{null} \) then \( Rm \)

else let \( h1 := \text{Rnext} \ M \ h \) in

let \( M1 := \text{freec} \ M \ h \) in \((M1, h1)\)

else let \( x_1 := \text{Rnext}_1 \ Rm \ H \ x \) in

if \( \text{eqd} \ Addr \ x_1 \ \text{null} \) then \( Rm \)

else let \( M2 := \text{mutc} \ M \ x_1 \) (modnext (loadc M x_1) \((\text{Rnext} \ M \ x))\) in

let \( M3 := \text{freec} \ M2 \ x \) in \((M3, h)\).

Definition \( \text{prec RD} \ Rm \ x := \)

forall \( k, \text{Rexd} \ Rm \ x \rightarrow RA \ (\text{fst} \ Rm) \ k \ x = x /\ RA_1 (\text{fst} \ Rm) \ k \ x = x. \)

**Example: Deletion of an isolated dart**

![Image of dart deletion example](image.png)

**Properties**

Correct behavior w.r.t. the observers \( \text{Rexd}, RA, RA_1, \) with proofs inherited from the general orbits: mutation and deletion properties
Observational Equivalence (I)

**Abstraction function (morphism): Abs**

Abs sends any hypermap representation which is *built by using* $RV, RI, RL, RD$ onto an abstract hypermap which is *built by using* $V, I, L$.

**Properties**

Abs *correctly carries* the observations by $Rexd, RA, RA_1$ onto the observations by $exd, A, A_1$. 
Observational Equivalence (II)

**Representation function (morphism):** \( \text{Rep} \)

\( \text{Rep} \) sends any abstract hypermap which is **built by using** \( V, I, L \) with \textit{darts generated by successive uses of} \( \text{adgen} \) onto a hypermap representation which is **built by using** \( RV, RI, RL \).

**Properties**

\( \text{Rep} \) **correctly carries** the observations by \( \text{exd}, A, A_1 \) onto the observations by \( \text{Rexd}, RA, RA_1 \).
Comments

- This appendix contains a C operational program for the concrete types, data structures and functions which correspond to the hypermap linked Coq representation.
- It is obtained by a direct translation where the memory is a global implicit object, memory variables are removed, addresses are pointers on cells, and $R_m$ is identified to $h$.
- A run on a test game needs a simple wrapping in ad hoc types and functions to refer darts in play, e.g. by integers, and to traverse the data structures.
Listing (I)

/* Programming in C the hypermap Coq representation */

#define MALLOC(t) ((t *) malloc(sizeof(t)))
#define null NULL

typedef enum {zero, one} dim;

typedef struct scell {
    struct scell * s[2];
    struct scell * p[2];
    struct scell * next;
} cell, * Addr, * Rhmap;

cell mkcell (Addr s[], Addr p[], Addr n) {
    cell c; int k;
    for(k=0;k<2;k++) {c.s[k] = s[k]; c.p[k] = p[k];}
    c.next := n;
    return c;
}
Listing (II)

(* CRm:CRhmap Rm is an inductive predicate stating that Rm is
created exclusively by using RV, RI, RL and RD: *)

Fixpoint Abs(Rm: Rhmap)(CRm:CRhmap Rm) {struct CRm}: fmap :=
  match CRm with ...

  cell mods(cell c, dim k, Addr m) { c.s[k] = m; return c; }

  cell modp(cell c, dim k, Addr m) { c.p[k] = m; return c; }

  cell modnext(cell c, Addr m) { c.next = m; return c; }

  cell ficell(Addr x) {
    cell c; int k;
    for(k=0;k<2;k++) { c.s[k] = c.p[k] = x; }
    c.next = null;
    return c;
  }

  cell initcell() {
    cell c; int k;
    for(k=0;k<2;k++) { c.s[k] = c.p[k] = null; }
    c.next = null;
    return c;
  }

J.-F. Dufourd (Strasbourg, ICUBE)
Listing (III)

cell load(Addr z) { return *z; }

void mut(Addr z, cell c) { *z = c; }

Addr alloc() {
    Addr x = MALLOC(cell);
    *x = initcell();
    return x;
}

/* free (z:Addr) BUILT-IN */

Addr Rnext (Addr z) { return z->next; }

Addr RA (dim k, Addr z) { return z->s[k]; }

Addr RA_1 (dim k, Addr z) { return z->p[k]; }
```c
Rhmap RV() { return null; }

Rhmap RI(Rhmap Rm) {
    Addr x = alloc();
    mut(x, (modnext(ficell(x), Rm)));
    return x;
}

Rhmap RL(Rhmap Rm, dim k, Addr x, Addr y) {
    Addr xk = RA(k, x);
    Addr y_k = RA_1(k, y);
    mut(x, (mods(load(x), k, y)));
    mut(y, (modp(load(y), k, x)));
    mut(y_k, (mods(load(y_k), k, x_k)));
    mut(xk, (modp(load(xk), k, y_k)));
    return Rm;
}
```
Listing (V)

Addr Rnext_1(Rhmap Rm, Addr x) {
    if(Rnext(Rm) == x) return Rm;
    return Rnext_1(Rnext(Rm), x);
}

Rhmap RD(Rhmap Rm, Addr x) {
    Addr h1, x_1;
    if (Rm == null || x == null) return Rm;
    if (Rm == x)
    {
        h1 = Rnext(Rm);
        free (Rm);
        return h1;
    }
    x_1 = Rnext_1(Rm,x);
    if (x_1 == null) return Rm;
    mut(x_1, (modnext (load(x_1), Rnext(x))));
    free(x);
    return Rm;
}
Related Work

Topics

- Static proofs of programs
  Floyd, Hoare, Reynolds, O’Hearn...
- Inductive types and algebraic specifications
  Bornat, Mehta-Nipkow, Marché, Conway-Barrett, Berdine et al..., Guttag, Goguen, Wirsing...
- Models of memory and of programming
  Leroy-Blazy, Chlipala...
- Separation and collision
  Burstall, Bornat, Reynolds, O’Hearn, Enea et al...
- Specification and implementation of hypermaps
  Cori, Gonthier, Dufourd, Bertrand, Bertot...
- Dedicated proof systems
  Malecha-Morrisett, Chlipala et al., Filliâtre...
Conclusion

Summary

- Interest for *libraries* with complex data structures
- Point of view of an *algebraic specifier*
- Exclusive use of a *higher-order logic (CIC)*: no Hoare logic, no Separation logic
- Intensive use of a *generic orbit library*: to handle arrays, singly- or doubly-linked lists, linear or circular, possibly nested
- *Coq development* for this study: 9,000 lines
  (with the memory model, but without the orbits, 60 definitions, 630 lemmas and theorems)
Conclusion

Future work

- Generalize *orbits* to *multiple functions*: to deal with trees, forests and general graphs.
- Connect *orbits* with *Separation logic*: orbits help to state and solve collision and separation problems.
- Connect *orbits* with *proof platforms*: Why3, Frama-C, Ynot, Bedrock...
- **Compile** the "imperative Coq fragment" to C.
- Develop other case studies with *complex data and algorithms*, particularly in computational geometry (Example: Delaunay / Voronoi diagrams in 3D).

Thank you for your attention!