Combining formalization and computation in Coq: a case-study with algebraic structures[1]

César Domínguez

Universidad de La Rioja

Algebraic computing, soft computing, and program verification

Castro Urdiales, April 2010

[1] Partially supported by Ministerio de Ciencia e Innovación, project MTM2009-13842-C02-01, and by European Commission FP7, STREP project ForMath.

1. Introduction.
2. A formalization in Coq of a hierarchy of data structures.
3. Some proofs and some instances.
4. Computing with instances in Coq.
5. Conclusions and further work.

Introduction

- Objective: To formalize some algorithms implemented in Kenzo.

Introduction

- Objective: To formalize some algorithms implemented in Kenzo.
- First milestone: mechanized proof of the Basic Perturbation Lemma in Isabelle/HOL.

Introduction

- Objective: To formalize some algorithms implemented in Kenzo.
- First milestone: mechanized proof of the Basic Perturbation Lemma in Isabelle/HOL.
- An approach to formalize this result in Type Theory in an abstract context (using Category theory concepts) is being carried out by T. Coquand and A. Spiwack.

Introduction

- Objective: To formalize some algorithms implemented in Kenzo.
- First milestone: mechanized proof of the Basic Perturbation Lemma in Isabelle/HOL.
- An approach to formalize this result in Type Theory in an abstract context (using Category theory concepts) is being carried out by T. Coquand and A. Spiwack.
- C. Domínguez, Julio Rubio
  The effective homology of bicomplexes, formalized in Coq

Introduction

- Objective: To formalize some algorithms implemented in Kenzo.
- First milestone: mechanized proof of the Basic Perturbation Lemma in Isabelle/HOL.
- An approach to formalize this result in Type Theory in an abstract context (using Category theory concepts) is being carried out by T. Coquand and A. Spiwack.
- C. Domínguez, Julio Rubio
  The effective homology of bicomplexes, formalized in Coq
- A hierarchy of algebraic (graded and infinite dimensional) data structures are required.

Introduction

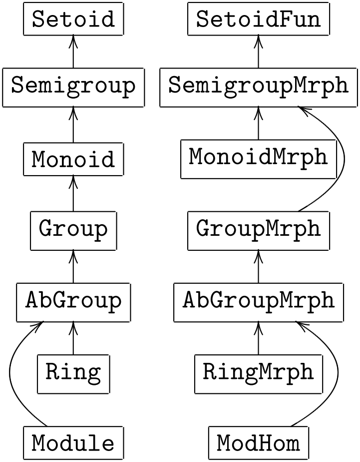
- Objective: To formalize some algorithms implemented in Kenzo.
- First milestone: mechanized proof of the Basic Perturbation Lemma in Isabelle/HOL.
- An approach to formalize this result in Type Theory in an abstract context (using Category theory concepts) is being carried out by T. Coquand and A. Spiwack.
- C. Domínguez, Julio Rubio
  The effective homology of bicomplexes, formalized in Coq
- A hierarchy of algebraic (graded and infinite dimensional) data structures are required.
- A sound and useful working representation in Coq: providing instances and proving theorems.

Introduction

- Objective: To formalize some algorithms implemented in Kenzo.
- First milestone: mechanized proof of the Basic Perturbation Lemma in Isabelle/HOL.
- An approach to formalize this result in Type Theory in an abstract context (using Category theory concepts) is being carried out by T. Coquand and A. Spiwack.
- C. Domínguez, Julio Rubio
  The effective homology of bicomplexes, formalized in Coq
- A hierarchy of algebraic (graded and infinite dimensional) data structures are required.
- A sound and useful working representation in Coq: providing instances and proving theorems.
- Constructive type theory allows proofs of computable terms.
  Idea: test and then formalize on particular instances.

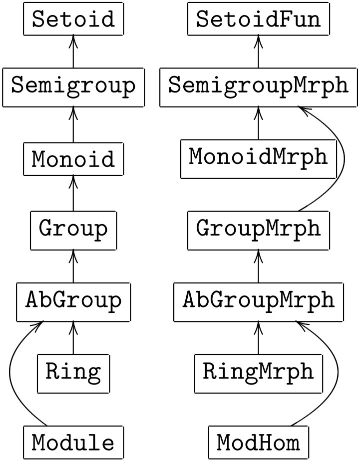
Basic algebraic structures from the CoRN repository

Formalization built on CoRN structures (in a simpler setting: setoids without apartness).



Basic algebraic structures from the CoRN repository

Formalization built on CoRN structures (in a simpler setting: setoids without apartness).



A new formulation of these structures in now in process.

Definitions for (constructive) free structures

Definitions for (constructive) free structures

- Let R be a ring. Given a set B, we consider the *free R-module generated by B*, denoted by $R[B]$.
  Its Coq formalization follows Algebra contribution by L. Pottier.

Definitions for (constructive) free structures

- Let R be a ring. Given a set B, we consider the *free R-module generated by B*, denoted by $R[B]$.
  Its Coq formalization follows Algebra contribution by L. Pottier.
- A *free* R-module M is a module where an explicit isomorphism is known between M and $R[B]$. The set of generators B is also explicitly given.

Definitions for (constructive) free structures

- Let R be a ring. Given a set B, we consider the *free R-module generated by B*, denoted by $R[B]$.
  Its Coq formalization follows Algebra contribution by L. Pottier.
- A *free* R-module M is a module where an explicit isomorphism is known between M and $R[B]$. The set of generators B is also explicitly given.
- A graded free R-module is a family of free R-modules $\{M_i\}_{i \in \mathbb{Z}}$.

Definitions for (constructive) free structures

- Let R be a ring. Given a set B, we consider the *free R-module generated by B*, denoted by $R[B]$.
  Its Coq formalization follows Algebra contribution by L. Pottier.
- A *free* R-module M is a module where an explicit isomorphism is known between M and $R[B]$. The set of generators B is also explicitly given.
- A graded free R-module is a family of free R-modules $\{M_i\}_{i\in\mathbb{Z}}$.
- A *chain complex* is a pair $(\{M_i\}_{i\in\mathbb{Z}}, \{d_i\}_{i\in\mathbb{Z}})$ where $\{M_i\}_{i\in\mathbb{Z}}$ is graded free module and $\{d_i\colon M_{i+1}\to M_i\}_{i\in\mathbb{Z}}$ is a family of module morphisms, called *differential operator*, such that $d_i\circ d_{i+1}=0$ for all $n\in\mathbb{Z}$.

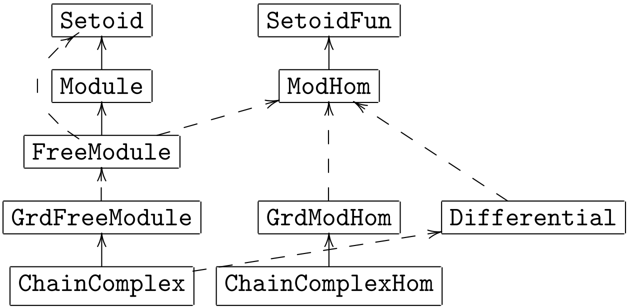
Chain complexes in Coq

- Given a ring `R: Ring`, a graded module can be formalized in Coq with the following type: `Z -> FreeModule R`.

Chain complexes in Coq

- Given a ring `R: Ring`, a graded module can be formalized in Coq with the following type: `Z -> FreeModule R`.
-
```
Record ChainComplex: Type:=
 {GrdMod:> Z -> FreeModule R;
  Diff: forall i: Z,
      ModHom (R:=R) (GrdMod (S i)) (GrdMod i);
  NilpotenceDiff: forall i: Z,
       (Nilpotence (Diff i)(Diff (S i))}.
```
- where the differential (nilpotence) property is defined by
```
Nilpotence(g: ModHom B C)(f: ModHom A B):=
forall a: A, ((g[oh]f) a)[=]Zero.
```

A hierarchy of data structures



Reductions

A *reduction* is a 5-tuple (TCC, BCC, f, g, h)

$$\begin{tikzcd} TCC \arrow[loop left, "h"] \arrow[r, bend left, "f"] & BCC \arrow[l, bend left, "g"] \end{tikzcd}$$

where $TCC = (M, d)$ and $BCC = (M', d')$ are chain complexes (named *top* and *bottom* chain complex), $f\colon TCC \to BCC$ and $g\colon BCC \to TCC$ are chain morphisms, $h = (h_i\colon M_i \to M_{i+1})_{i\in\mathbb{Z}}$ is a family of module morphisms (called *homotopy operator*), which satisfy the following properties for all $i \in \mathbb{Z}$:

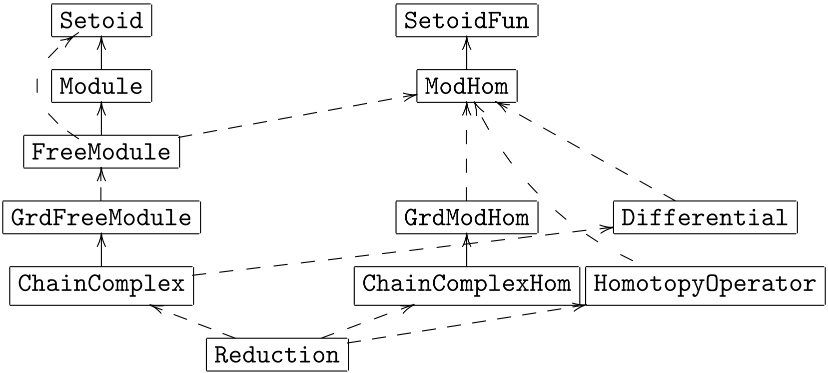
1. $f_i \circ g_i = id_{M'_i}$
2. $d_{i+1} \circ h_{i+1} + h_i \circ d_i + g_{i+1} \circ f_{i+1} = id_{M_{i+1}}$
3. $f_{i+1} \circ h_i = 0$
4. $h_i \circ g_i = 0$
5. $h_{i+1} \circ h_i = 0$

Reductions in Coq

- In Coq, this concept is again formalized as a record:

```
Record Reduction:Type:=
 {topCC: ChainComplex R;
  bottomCC: ChainComplex R;
  f_t_b: ChainComplex_hom topCC bottomCC;
  g_b_t: ChainComplex_hom bottomCC topCC;
  h_t_t: HomotopyOperator topCC;
  rp1: forall (i: Z)(a:(bottomCC i)),
       ((f_t_b i)[oh](g_b_t i))a[=]a;
    ...
```

A hierarchy of data structures



Definitions for (constructive) effective homology

- Given a natural number $k \in \mathbb{N}$, let us denote $FS(k)$ the (finite) set $\{0, \ldots, k-1\}$.

Definitions for (constructive) effective homology

- Given a natural number $k \in \mathbb{N}$, let us denote $FS(k)$ the (finite) set $\{0, \ldots, k-1\}$.
- A set B is *finite* if it is endowed with a natural $k \in \mathbb{N}$ and an explicit bijection $\psi : B \to FS(k)$ with an explicit inverse $\psi^{-1} : FS(k) \to B$.

Definitions for (constructive) effective homology

- Given a natural number $k \in \mathbb{N}$, let us denote $FS(k)$ the (finite) set $\{0, \ldots, k-1\}$.
- A set B is *finite* if it is endowed with a natural $k \in \mathbb{N}$ and an explicit bijection $\psi : B \to FS(k)$ with an explicit inverse $\psi^{-1} : FS(k) \to B$.
- A *free* R-module M is *of finite type* if its set of generators is finite.

Definitions for (constructive) effective homology

- Given a natural number $k \in \mathbb{N}$, let us denote $FS(k)$ the (finite) set $\{0, \ldots, k-1\}$.
- A set B is *finite* if it is endowed with a natural $k \in \mathbb{N}$ and an explicit bijection $\psi : B \to FS(k)$ with an explicit inverse $\psi^{-1} : FS(k) \to B$.
- A *free* R-module M is *of finite type* if its set of generators is finite.
- These definitions extend and apply naturally to graded modules, chain complexes, chain morphisms, ...

Definitions for (constructive) effective homology

- Given a natural number $k \in \mathbb{N}$, let us denote $FS(k)$ the (finite) set $\{0, \ldots, k-1\}$.
- A set B is *finite* if it is endowed with a natural $k \in \mathbb{N}$ and an explicit bijection $\psi : B \rightarrow FS(k)$ with an explicit inverse $\psi^{-1} : FS(k) \rightarrow B$.
- A *free* R-module M is *of finite type* if its set of generators is finite.
- These definitions extend and apply naturally to graded modules, chain complexes, chain morphisms, ...
- A chain complex CC with effective homology is a reduction (CC, FCC, f, g, h) where FCC is a *free of finite type* chain complex.

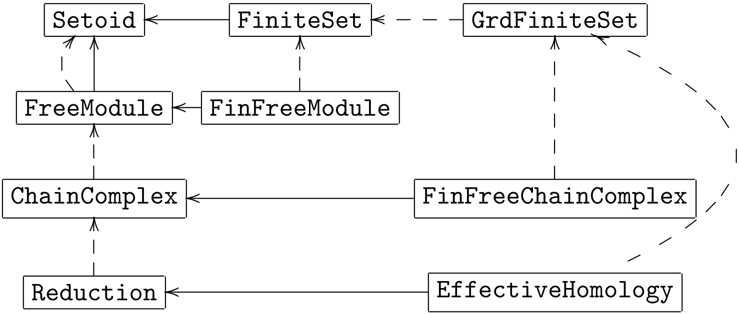
Free of finite type modules

```
Record FinFreeModule: Type  :=
 {FFM :> FreeModule R;
  Finite_SetoidFFM: Finite_Set;
  equal_FFM: (is_FFModule FFM Finite_SetoidFFM)
 }.

Definition is_FFModule(FM:FreeModule R)(X:Finite_Set):=
                      SGen(FM) = X.
```

`SGen` = Set of generators of the FreeModule.

A hierarchy of data structures



Mapping cones

Definition

Given a pair of chain complexes $CC = ((M_i)_{i\in\mathbb{Z}}, (d_i)_{i\in\mathbb{Z}})$ and $CC' = ((M'_i)_{i\in\mathbb{Z}}, (d'_i)_{i\in\mathbb{Z}})$ and a chain complex morphism $\alpha\colon CC \to CC'$, the *cone* of α, denoted by $Cone(\alpha)$, is a chain complex $((M''_i)_{i\in\mathbb{Z}}, (d''_i)_{i\in\mathbb{Z}})$ such that, for each $i \in \mathbb{Z}$, $M''_i = M_i \oplus M'_{i+1}$ and $d''_i(x, x') = (-d_i(x), d'_{i+1}(x') + \alpha_{i+1}(x))$ for any $x \in M_{i+1}$ and $x' \in M'_{i+2}$.

$$
\begin{array}{ccccccccccccc}
\cdots & \xleftarrow{d_{-3}} & M_{-2} & \xleftarrow{d_{-2}} & M_{-1} & \xleftarrow{d_1} & M_0 & \xleftarrow{d_0} & M_1 & \xleftarrow{d_1} & M_2 & \xleftarrow{d_2} & \cdots \\
\cdots & \leftarrow d''_{-3} - & \times & \leftarrow d''_{-2} - & \times & \leftarrow d''_{-1} - & \times & \leftarrow d''_0 - & \times & \leftarrow d''_1 - & \times & \leftarrow d''_2 - & \cdots \\
\cdots & \xleftarrow[d_{-2}]{} & M'_{-1} & \xleftarrow[d_{-1}]{} & M'_0 & \xleftarrow[d_2]{} & M'_1 & \xleftarrow[d_1]{} & M'_2 & \xleftarrow[d_2]{} & M'_3 & \xleftarrow[d_4]{} & \cdots
\end{array}
$$

Mapping cones

Definition

Given a pair of chain complexes $CC = ((M_i)_{i\in\mathbb{Z}}, (d_i)_{i\in\mathbb{Z}})$ and $CC' = ((M'_i)_{i\in\mathbb{Z}}, (d'_i)_{i\in\mathbb{Z}})$ and a chain complex morphism $\alpha\colon CC \to CC'$, the *cone* of α, denoted by $Cone(\alpha)$, is a chain complex $((M''_i)_{i\in\mathbb{Z}}, (d''_i)_{i\in\mathbb{Z}})$ such that, for each $i \in \mathbb{Z}$, $M''_i = M_i \oplus M'_{i+1}$ and $d''_i(x, x') = (-d_i(x), d'_{i+1}(x') + \alpha_{i+1}(x))$ for any $x \in M_{i+1}$ and $x' \in M'_{i+2}$.

$$
\begin{array}{ccccccccccccc}
\cdots & \xleftarrow{d_{-3}} & M_{-2} & \xleftarrow{d_{-2}} & M_{-1} & \xleftarrow{d_1} & M_0 & \xleftarrow{d_0} & M_1 & \xleftarrow{d_1} & M_2 & \xleftarrow{d_2} & \cdots \\
\cdots & \leftarrow d''_{-3} - & \times & \leftarrow d''_{-2} - & \times & \leftarrow d''_{-1} - & \times & \leftarrow d''_{0} - & \times & \leftarrow d''_{1} - & \times & \leftarrow d''_{2} - & \cdots \\
\cdots & \xleftarrow[d_{-2}]{} & M'_{-1} & \xleftarrow[d_{-1}]{} & M'_0 & \xleftarrow[d_2]{} & M'_1 & \xleftarrow[d_1]{} & M'_2 & \xleftarrow[d_2]{} & M'_3 & \xleftarrow[d_4]{} & \cdots
\end{array}
$$

In Coq:

```
Definition ConeDiffGround:=fun(i:Z)(ab:(ConeGround (i+1)))=>
  ([--](Diff CC1 i(fst ab)),
  ((Diff CC0(i+1))(snd ab)[+]f(i+1)(fst ab))).
```

Effective homology of a mapping cone

Theorem

Given two reductions $r = (TCC, BCC, f, g, h)$ *and* $r' = (TCC', BCC', f', g', h')$ *and a chain morphism* $\alpha \colon TCC \to TCC'$, *it is possible to define a reduction* $r'' = (Cone(\alpha), BCC'', f'', g'', h'')$ *with* $Cone(\alpha)$ *as top chain complex and:*

- $BCC'' = Cone(\alpha')$ *with* $\alpha' \colon BCC \to BCC'$ *defined by* $\alpha' = f' \circ \alpha \circ g$
- $f'' = (f, f' \circ \alpha \circ h + f')$, $g'' = (g, -h' \circ \alpha \circ g + g')$, $h'' = (-h, h' \circ \alpha \circ h + h')$

$$\begin{tikzcd}
TCC \arrow[loop above, "h"] \arrow[r, shift left, "f"] \arrow[d, "\alpha"'] & BCC \arrow[l, shift left, "g"] \arrow[d, dashed, "\alpha'"'] \\
TCC' \arrow[loop below, "h'"] \arrow[r, shift left, "f'"] & BCC' \arrow[l, shift left, "g'"]
\end{tikzcd}$$

Besides, if TCC *and* TCC' *are objects with effective homology through* r *and* r', *then* $Cone(\alpha)$ *is and object with effective homology through* r''.

Effective homology of a cone, in Coq

- Given two reductions `r1 r2: Reduction R`, and a chain morphism between their top chain complexes `alpha: ChainComplex_hom (topCC r1) (topCC r2)`,

Effective homology of a cone, in Coq

- Given two reductions `r1 r2: Reduction R`, and a chain morphism between their top chain complexes `alpha: ChainComplex_hom (topCC r1) (topCC r2)`,
- Define a chain morphism `alpha'` between the bottom chain complexes through the function

```
Definition alpha'':= fun i: Z =>
(f_t_b r2 i) [oh] (alpha i) [oh] (g_b_t r1 i).
```

Effective homology of a cone, in Coq

- Given two reductions `r1 r2: Reduction R`, and a chain morphism between their top chain complexes
  `alpha: ChainComplex_hom (topCC r1) (topCC r2)`,
- Define a chain morphism `alpha'` between the bottom chain complexes through the function
  ```
  Definition alpha'':= fun i: Z =>
  (f_t_b r2 i) [oh] (alpha i) [oh] (g_b_t r1 i).
  ```
- Then we build a reduction between `Cone(alpha)` and `Cone(alpha')`.

Effective homology of a cone, in Coq

- Given two reductions `r1 r2: Reduction R`, and a chain morphism between their top chain complexes `alpha: ChainComplex_hom (topCC r1) (topCC r2)`,
- Define a chain morphism `alpha'` between the bottom chain complexes through the function
  `Definition alpha'':= fun i: Z =>`
  `(f_t_b r2 i) [oh] (alpha i) [oh] (g_b_t r1 i).`
- Then we build a reduction between `Cone(alpha)` and `Cone(alpha')`.
- For instance, the first chain morphism of the reduction is:

```
Definition f_cone_reductionGround:
 forall i: Z, (Cone alpha)i -> (Cone alpha')i:=
  fun (i: Z)(ab: (Cone alpha)i) => ((f_t_b r1 i)(fst ab),
   (((f_t_b r2 (i+1)) [oh] (alpha (i+1)) [oh]
    (h_t_t r1 i))(fst ab)) [+] (f_t_b r2 (i+1))(snd ab)).
```

Instances of the structures, in Coq

- Finite Type:
  - Null free module

Instances of the structures, in Coq

- Finite Type:
  - Null free module
  - Integer numbers

Instances of the structures, in Coq

- Finite Type:
  - Null free module
  - Integer numbers
  - The chain complex $FCC^{(1)}$:

$$
\begin{tikzcd}[row sep=small]
\cdots & \mathbb{Z} \arrow[l, "0"'] & \mathbb{Z} \arrow[l, "\times 2"'] & \mathbb{Z} \arrow[l, "0"'] & \mathbb{Z} \arrow[l, "\times 2"'] & \mathbb{Z} \arrow[l, "0"'] & \cdots \arrow[l, "\times 2"'] \\
\text{degree} & \text{-2} & \text{-1} & 0 & 1 & 2 &
\end{tikzcd}
$$

Instances of the structures, in Coq

- Finite Type:
  - Null free module
  - Integer numbers
  - The chain complex $FCC^{(1)}$:

$$
\begin{tikzcd}
\cdots & \mathbb{Z} \arrow[l, "0"'] & \mathbb{Z} \arrow[l, "\times 2"'] & \mathbb{Z} \arrow[l, "0"'] & \mathbb{Z} \arrow[l, "\times 2"'] & \mathbb{Z} \arrow[l, "0"'] & \cdots \arrow[l, "\times 2"'] \\
\text{degree} & \text{-2} & \text{-1} & 0 & 1 & 2 &
\end{tikzcd}
$$

- Infinite Type:
  - Module $\mathbb{Z}[\mathbb{N}]$

Instances of the structures, in Coq

- Finite Type:
  - Null free module
  - Integer numbers
  - The chain complex $FCC^{(1)}$:

$$
\begin{array}{ccccccccccccc}
\cdots & \xleftarrow{0} & \mathbb{Z} & \xleftarrow{\times 2} & \mathbb{Z} & \xleftarrow{0} & \mathbb{Z} & \xleftarrow{\times 2} & \mathbb{Z} & \xleftarrow{0} & \mathbb{Z} & \xleftarrow{\times 2} & \cdots \\
\text{degree} & & \text{-2} & & \text{-1} & & 0 & & 1 & & 2 & &
\end{array}
$$

- Infinite Type:
  - Module $\mathbb{Z}[\mathbb{N}]$
  - The chain complex $CC^{(2)}$:

$$
\begin{array}{ccc}
\mathbb{Z}[\mathbb{N}] & \xleftarrow[(d^{(2)})_i]{\text{i even}} & \mathbb{Z}[\mathbb{N}] \\
x_0 & \longleftarrow\!\shortmid & x_0 \\
0 & \longleftarrow\!\shortmid & x_1 \\
x_2 & \longleftarrow\!\shortmid & x_2 \\
0 & \longleftarrow\!\shortmid & x_3 \\
\cdots & & \cdots
\end{array}
\qquad
\begin{array}{ccc}
\mathbb{Z}[\mathbb{N}] & \xleftarrow[(d^{(2)})_i]{\text{i odd}} & \mathbb{Z}[\mathbb{N}] \\
0 & \longleftarrow\!\shortmid & x_0 \\
x_1 & \longleftarrow\!\shortmid & x_1 \\
0 & \longleftarrow\!\shortmid & x_2 \\
x_3 & \longleftarrow\!\shortmid & x_3 \\
\cdots & & \cdots
\end{array}
$$

Instances of effective homologies, in Coq

- $CC^{(1)}$ is $FCC^{(1)}$ without the finiteness condition:

$$\begin{tikzcd} CC^{(1)} \arrow[loop above, "0"] \arrow[r, bend left, "id"] & FCC^{(1)} \arrow[l, bend left, "id"] \end{tikzcd}$$

Instances of effective homologies, in Coq

- $CC^{(1)}$ is $FCC^{(1)}$ without the finiteness condition:

$$\begin{tikzcd} CC^{(1)} \arrow[loop above, "0"] \arrow[r, bend left, "id"] & FCC^{(1)} \arrow[l, bend left, "id"] \end{tikzcd}$$

- $h^{(2)}$ defined as the $d^{(2)}$ differential:

$$\begin{tikzcd} CC^{(1)} \oplus CC^{(2)} \arrow[loop above, "{(0,h^{(2)})}"] \arrow[r, bend left, "\pi_1"] & FCC^{(1)} \arrow[l, bend left, "{(id,0)}"] \end{tikzcd}$$

Instances of effective homologies, in Coq

$$
\begin{tikzcd}[column sep=huge]
CC^{(1)} \oplus CC^{(2)} \arrow[loop above, "{(0,h^{(2)})}"] \arrow[r, bend left=10, "\pi_1"] \arrow[d, "\pi_1"'] & FCC^{(1)} \arrow[l, bend left=10, "{(id,0)}"] \arrow[d, dashed, "\alpha'"'] \\
CC^{(1)} \arrow[loop below, "0"] \arrow[r, bend left=10, "id"] & FCC^{(1)} \arrow[l, bend left=10, "id"]
\end{tikzcd}
$$

Instances of effective homologies, in Coq

$$\begin{tikzcd}
CC^{(1)} \oplus CC^{(2)} \arrow[loop above, "{(0,h^{(2)})}"] \arrow[rr, shift left, "\pi_1"] \arrow[d, "\pi_1"'] & & FCC^{(1)} \arrow[ll, shift left, "{(id,0)}"] \arrow[d, dashed, "\alpha'"'] \\
CC^{(1)} \arrow[loop below, "0"] \arrow[rr, shift left, "id"] & & FCC^{(1)} \arrow[ll, shift left, "id"]
\end{tikzcd}$$

Effective homology of the cone of π_1

$$\begin{tikzcd}
Cone(\pi_1) \arrow[loop above, "h^{Ex}"] \arrow[rr, shift left, "f^{Ex}"] & & Cone(\alpha') \arrow[ll, shift left, "g^{Ex}"]
\end{tikzcd}$$

Computing with instances in Coq

Looking for a contracting homotopy in $Cone(\alpha')$, *i.e.* a $h : Cone(\alpha') \rightarrow Cone(\alpha')$ such that $h \circ h = 0$ and $d \circ h + h \circ d = id$ in

$$\begin{tikzcd} Cone(\pi_1) \arrow[loop above, "h^{Ex}"] \arrow[rr, bend left=10, "f^{Ex}"] & & Cone(\alpha') \arrow[loop above, "\color{blue}h"] \arrow[ll, bend left=10, "g^{Ex}"] \end{tikzcd}$$

Computing with instances in Coq

Looking for a contracting homotopy in $Cone(\alpha')$, *i.e.* a $h : Cone(\alpha') \to Cone(\alpha')$ such that $h \circ h = 0$ and $d \circ h + h \circ d = id$ in

$$\begin{tikzcd} Cone(\pi_1) \arrow[loop above, "h^{Ex}"] \arrow[r, bend left=10, "f^{Ex}"] & Cone(\alpha') \arrow[loop above, "h"] \arrow[l, bend left=10, "g^{Ex}"] \end{tikzcd}$$

- Candidates:
  - $h1 = (h1_i)_{i \in \mathbb{Z}}$, such that $h1_i(a,b) := (0,a)$, for all $i \in \mathbb{Z}$
  - $h2 = (h2_i)_{i \in \mathbb{Z}}$, such that $h2_i(a,b) := (b,0)$, for all $i \in \mathbb{Z}$

Computing with finite instances, in Coq

- Candidates in Coq:
  - `Definition h1:=fun(i:Z)(c:bottomCC Example i)=>(0,fst c)`
  - `Definition h2:=fun(i:Z)(c:bottomCC Example i)=>(snd c,0)`

Computing with finite instances, in Coq

- Candidates in Coq:
  - `Definition h1:=fun(i:Z)(c:bottomCC Example i)=>(0,fst c)`
  - `Definition h2:=fun(i:Z)(c:bottomCC Example i)=>(snd c,0)`
- Computing in Coq ($d \circ h + h \circ d = id$) with *finite* structures:
  - ```
    Eval vm_compute in
    (((Diff (bottomCC Example) 2)[oh](h1 2))[+h]
     ((h1 1)[oh](Diff(bottomCC Example) 1)))(5, 7).
    ```

    resulting in: `=(0, 0):bottomCC Example 2`
    ```

Computing with finite instances, in Coq

- Candidates in Coq:
  - `Definition h1:=fun(i:Z)(c:bottomCC Example i)=>(0,fst c)`
  - `Definition h2:=fun(i:Z)(c:bottomCC Example i)=>(snd c,0)`
- Computing in Coq ($d \circ h + h \circ d = id$) with *finite* structures:
  - ```
    Eval vm_compute in
    (((Diff (bottomCC Example) 2)[oh](h1 2))[+h]
     ((h1 1)[oh](Diff(bottomCC Example) 1)))(5, 7).
    ```
    resulting in: `=(0, 0):bottomCC Example 2`
  - ```
    Eval vm_compute in
    (((Diff (bottomCC Example) 2)[oh](h2 2))[+h]
     ((h2 1)[oh](Diff(bottomCC Example) 1)))(5, 7).
    ```
    resulting in: `=(5, 7):bottomCC Example 2`

Computing with infinite instances, in Coq

$$\begin{tikzcd}
Cone(\pi_1) \arrow[loop above, "h^{Ex}"] \arrow[loop below, "h^{Ex} + g^{Ex} \circ h2 \circ f^{Ex}"] \arrow[r, bend left, "f^{Ex}"] & Cone(\alpha') \arrow[loop above, "h2"] \arrow[l, bend left, "g^{Ex}"]
\end{tikzcd}$$

Computing with infinite instances, in Coq

$$
\begin{tikzcd}
Cone(\pi_1) \arrow[loop above, "h^{Ex}"] \arrow[loop below, "h^{Ex} + g^{Ex} \circ {\color{blue}h2} \circ f^{Ex}"'] \arrow[r, bend left=10, "f^{Ex}"] & Cone(\alpha') \arrow[loop above, "{\color{blue}h2}"] \arrow[l, bend left=10, "g^{Ex}"]
\end{tikzcd}
$$

- In Coq

```
Definition h_topCone:= fun n: Z => (h_t_t Example) n [+h]
  (((g_b_t Example) n) [oh] (h2 n) [oh] ((f_t_b Example) n)).
```

Computing with infinite instances, in Coq

$$
\begin{tikzcd}
\mathit{Cone}(\pi_1) \arrow[loop above, "h^{Ex}"] \arrow[loop below, "h^{Ex} + g^{Ex} \circ h2 \circ f^{Ex}"] \arrow[r, bend left, "f^{Ex}"] & \mathit{Cone}(\alpha') \arrow[l, bend left, "g^{Ex}"] \arrow[loop above, "h2"]
\end{tikzcd}
$$

- In Coq

```
Definition h_topCone:= fun n: Z => (h_t_t Example) n [+h]
  (((g_b_t Example) n) [oh] (h2 n) [oh] ((f_t_b Example) n)).
```

- Computing in Coq with *infinite* structures:
  (e represents the element $(7 * x_4 + 8 * x_0)$).

  - ```
    Eval vm_compute in
    (((Diff(topCC Example) 2)[oh](h_topCone 2))[+h]
     ((h_topCone 1)[oh]((Diff(topCC Example) 1))))(5, e, 3).
    ```

    resulting in an element equal (in the setoid) to `(5, e, 3)`.
    ```

Computing with infinite instances, in Coq

Undecidible problem in general

When working with chain complexes of infinite type, if an element x is a cycle (that is to say, $d_n(x) = 0$) and the chain complex has a contracting homotopy, then there exists an element z such that $d_{n+1}(z) = x$.

Computing with infinite instances, in Coq

Undecidible problem in general

When working with chain complexes of infinite type, if an element x is a cycle (that is to say, $d_n(x) = 0$) and the chain complex has a contracting homotopy, then there exists an element z such that $d_{n+1}(z) = x$.

- For a chain complex with a contracting homotopy it is possible to calculate that pre-image for some elements.
- Example `(-10, Inv e, 5)`

Computing with infinite instances, in Coq

Undecidible problem in general

When working with chain complexes of infinite type, if an element x is a cycle (that is to say, $d_n(x) = 0$) and the chain complex has a contracting homotopy, then there exists an element z such that $d_{n+1}(z) = x$.

- For a chain complex with a contracting homotopy it is possible to calculate that pre-image for some elements.
- Example `(-10, Inv e, 5)`
- Computing in Coq with *infinite* structures:

```
Eval vm_compute in (h_topCone 2)(-10, Inv e, 5).
```

resulting in an element equal to `(5, e, 0)`.

Computing with infinite instances, in Coq

Undecidible problem in general

When working with chain complexes of infinite type, if an element x is a cycle (that is to say, $d_n(x) = 0$) and the chain complex has a contracting homotopy, then there exists an element z such that $d_{n+1}(z) = x$.

- For a chain complex with a contracting homotopy it is possible to calculate that pre-image for some elements.
- Example `(-10, Inv e, 5)`
- Computing in Coq with *infinite* structures:

  ```
  Eval vm_compute in (h_topCone 2)(-10, Inv e, 5).
  ```

  resulting in an element equal to `(5, e, 0)`.

  ```
  Eval vm_compute in ((Diff(topCC Example) 2))(5, e, 0).
  ```

  resulting the required element `(-10, Inv e, 5)`.

Conclusions and further work

- Conclusions:
  - Algebraic Topology can be formalized in Coq.

Conclusions and further work

- Conclusions:
  - Algebraic Topology can be formalized in Coq.
  - In particular, we have obtained the formalization of a hierarchy of data structures in Algebraic Topology

Conclusions and further work

- Conclusions:
  - Algebraic Topology can be formalized in Coq.
  - In particular, we have obtained the formalization of a hierarchy of data structures in Algebraic Topology
  - We have provided some proofs and some instances of the structures.

Conclusions and further work

- Conclusions:
  - Algebraic Topology can be formalized in Coq.
  - In particular, we have obtained the formalization of a hierarchy of data structures in Algebraic Topology
  - We have provided some proofs and some instances of the structures.
  - The instances allow us to relate deduction and computing in Coq.

Conclusions and further work

- Conclusions:
  - Algebraic Topology can be formalized in Coq.
  - In particular, we have obtained the formalization of a hierarchy of data structures in Algebraic Topology
  - We have provided some proofs and some instances of the structures.
  - The instances allow us to relate deduction and computing in Coq.
- Further work:

Conclusions and further work

- Conclusions:
  - Algebraic Topology can be formalized in Coq.
  - In particular, we have obtained the formalization of a hierarchy of data structures in Algebraic Topology
  - We have provided some proofs and some instances of the structures.
  - The instances allow us to relate deduction and computing in Coq.
- Further work:
  - We are ready to rebuild our hierarchy using new formalization techniques in CoRN and/or ssreflect.

Conclusions and further work

- Conclusions:
  - Algebraic Topology can be formalized in Coq.
  - In particular, we have obtained the formalization of a hierarchy of data structures in Algebraic Topology
  - We have provided some proofs and some instances of the structures.
  - The instances allow us to relate deduction and computing in Coq.
- Further work:
  - We are ready to rebuild our hierarchy using new formalization techniques in CoRN and/or ssreflect.
  - Extracting programs to (Common) Lisp.