

Pushout.lisp

Jónathan Heras

Departamento de Matemáticas y Computación. Universidad de la Rioja.

1 Mathematical Aspects

Most of definition and theorems that we present in this section can be found in [3].

Definition 1 (Reduction) A *reduction* $\rho : \hat{C}_* \Longrightarrow C_*$ is a diagram:

$$\rho = \begin{array}{ccc} & \xrightarrow{h} & \\ & \hat{C}_* & \xrightarrow{f} C_* \\ & \xleftarrow{g} & \end{array}$$

where:

1. \hat{C}_* and C_* are chain-complexes.
2. f and g are chain complex morphisms.
3. h is a homotopy operator (degree +1).
4. These relations are satisfied:

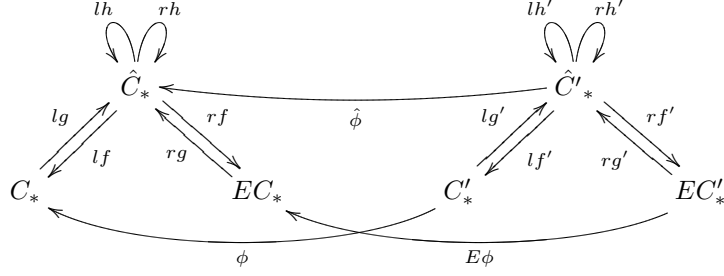
- (a) $fg = id_{C_*}$.
- (b) $gf + dh + hd = id_{\hat{C}_*}$.
- (c) $fh = hg = hh = 0$.

Definition 2 (Cone) Let C_* and D_* be two chain-complexes and $\phi : C_* \leftarrow D_*$ be a chain-complex morphism. Then the cone of ϕ denoted by $Cone(\phi)$ is the chain complex $Cone(\phi) = A_*$ defined as follows. First $A_n := C_n \oplus D_{n-1}$ (or $A_n := C_{n+1} \oplus D_n$); then the boundary operator is given by the matrix:

$$d_{A_*} = \begin{bmatrix} d_{C_*} & \phi \\ 0 & -d_{D_*} \end{bmatrix}$$

Theorem 1 (Cone Reduction Theorem) Let $\rho = (f, g, h) : \hat{C}_* \Longrightarrow D_*$ and $\rho' = (f', g', h') : \hat{C}'_* \Longrightarrow D'_*$ be two reductions and $\phi : C_* \leftarrow C'_*$ a chain complex morphism. Then these data define a canonical reduction:

$$\rho'' = (f'', g'', h'') : Cone(\phi) \Longrightarrow Cone(f\phi g') .$$



The proof of this theorem can be found in Page 57 of [3].

Theorem 2 (Cone Equivalence Theorem) Let $\phi : C_{*,EH} \leftarrow C'_{*,EH}$ be a chain complex morphism between two chain-complexes with effective homology. Then a general algorithm computes a version with effective homology $Cone(\phi)_{EH}$ of the cone.

Theorem 3 (SES Theorems) Let

$$0 \xleftarrow{0} A_* \xrightleftharpoons[j]{\sigma} B_* \xrightleftharpoons[i]{\rho} C_* \xleftarrow{0} 0$$

be an effective short exact sequence of chain-complexes. Then three general algorithms are available:

$$\begin{aligned} SES_1 &: (B_{*,EH}, C_{*,EH}) \mapsto A_{*,EH} \\ SES_2 &: (A_{*,EH}, C_{*,EH}) \mapsto B_{*,EH} \\ SES_3 &: (A_{*,EH}, B_{*,EH}) \mapsto C_{*,EH} \end{aligned}$$

The proof of these theorems can be found in Page 71 of [3].

Definition 3 (Pushout) Let X, Y and Z three simplicial sets and $f : X \rightarrow Y$ and $g : X \rightarrow Z$ two simplicial maps. Then the *pushout* of (f, g) is the disjoint union

$$(Y \coprod X \times I \coprod Z) / \sim$$

where the equivalence relation \sim is defined as follows. For every simplex $x \in X$, on one hand $(x \times 0)$ is identified to $f(x) \in Y$ and $(x \times 1)$ is identified to $g(x) \in Z$.

A more detailed description of pushouts can be found in [2].

Theorem 4 Let $EFHM(X), EFHM(Y), EFHM(Z)$, $f : X \rightarrow Y$ and $g : X \rightarrow Z$ we can compute $EFHM(P)$ where P is the pushout of f and g .

Proof.

We have the short exact sequence:

$$0 \xleftarrow{\quad} M \xrightleftharpoons{\quad} C(X \times I) \xrightleftharpoons{\quad} C(X \times \{0\}) \oplus C(X \times \{1\}) \xleftarrow{\quad} 0$$

where M is the chain complex coming from $X \times I$ but with the simplices of $X \times \{0\}$ and $X \times \{1\}$ cancelled.

In this case the functions i, j, σ and ρ are defined as follows:

$$\begin{aligned}
i : C(X \times \{0\}) \oplus C(X \times \{1\}) &\rightarrow C(X \times I) \\
x \times \{0\} &\mapsto x \times \{0\} \\
x \times \{1\} &\mapsto x \times \{1\} \\
\\
j : C(X \times I) &\rightarrow M \\
x \times \{a\} &\mapsto x \times \{a\} \quad \text{if } a \neq 0, 1 \\
x \times \{0\} &\mapsto nil \\
x \times \{1\} &\mapsto nil \\
\\
\sigma : M &\rightarrow C(X \times I) \\
x \times \{a\} &\mapsto x \times \{a\} \\
\\
\rho : C(X \times I) &\rightarrow C(X \times \{0\}) \oplus C(X \times \{1\}) \\
x \times \{a\} &\mapsto nil \quad \text{if } a \neq 0, 1 \\
x \times \{0\} &\mapsto x \times \{0\} \\
x \times \{1\} &\mapsto x \times \{1\}
\end{aligned}$$

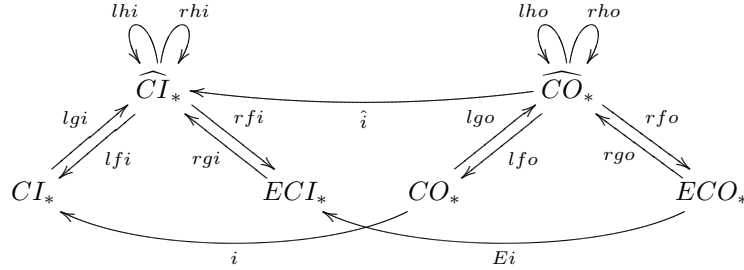
From the SES_1 Theorem we get the sequence

$$M \Leftarrow Cone(i) \Leftarrow Cone(\hat{i}) \Rightarrow Cone(Ei)$$

in the following way:

To simplify the notation, we are going to use CI for $C(X \times I)$ and CO for $C(X \times \{0\}) \oplus C(X \times \{1\})$

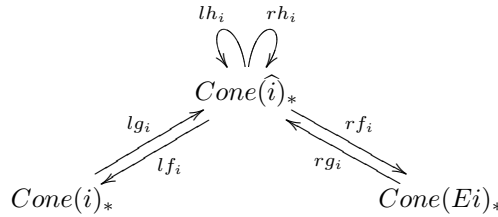
We have two equivalences $CI \Leftarrow \widehat{CI} \Rightarrow ECI$ and $CO \Leftarrow \widehat{CO} \Rightarrow ECO$ and the morphism $i : CI \leftarrow CO$.



The morphism i naturally induces “parallel” morphism $\hat{i} := (lgi)\phi(lfo) : \widehat{CI}_* \leftarrow \widehat{CO}_*$ and then $Ei := (rfi)(lgi)\phi(lfo)(rgo) : ECI_* \leftarrow ECO_*$.

Applying the Cone Equivalence Theorem, we get the equivalence

$$Cone(i) \Leftarrow Cone(\hat{i}) \Rightarrow Cone(Ei)$$



where

$$d_{Cone(i)_*} = \begin{bmatrix} d_{CI_*} & i \\ 0 & d_{CO_*} \end{bmatrix}, d_{Cone(\hat{i})_*} = \begin{bmatrix} d_{\widehat{CI}_*} & \hat{i} \\ 0 & d_{\widehat{CO}_*} \end{bmatrix}, d_{Cone(Ei)_*} = \begin{bmatrix} d_{ECI_*} & Ei \\ 0 & d_{ECO_*} \end{bmatrix}$$

$$lf_i = \begin{bmatrix} lfi & 0 \\ 0 & lfo \end{bmatrix}, lg_i = \begin{bmatrix} lgi & 0 \\ 0 & lgo \end{bmatrix}, lh_i = \begin{bmatrix} lhi & 0 \\ 0 & -lho \end{bmatrix},$$

$$rf_i = \begin{bmatrix} rfi & (rfi)\hat{i}(rho) \\ 0 & rfo \end{bmatrix}, rg_i = \begin{bmatrix} rgi & -(rhi)\hat{i}(rgo) \\ 0 & rgo \end{bmatrix}, rh_i = \begin{bmatrix} rhi & (rhi)\hat{i}(rho) \\ 0 & -rho \end{bmatrix}.$$

Now, if we apply the SES_1 theorem, we get:

$$M \Longleftarrow Cone(i) \Longleftarrow Cone(\hat{i}) \Longrightarrow Cone(Ei)$$

The values for $lf_i, lg_i, lh_i, rf_i, rg_i, rh_i$ where described previously and $fm = j$, $gm = \sigma - \rho d_{CI}\sigma$ and $h1 = \rho$.

We can now compose the two reductions $M \Longleftarrow Cone(i)$ and $Cone(i) \Longleftarrow Cone(\hat{i})$ and we obtain:

$$M \Longleftarrow Cone(\hat{i}) \Longrightarrow Cone(Ei)$$

where $lf'_i = fm \circ lf_i$, $lg'_i = lg_i \circ gm$ and $lh'_i = lh_i + lg_i \circ hm \circ lf_i$.

Then, if we know $EFHM(CX)$ and $EFHM(C(X \times I))$ we can obtain $EFHM(M)$

Let us consider now, the short exact sequence:

$$0 \longleftarrow M \Longleftrightarrow CP \Longleftrightarrow CY \oplus CZ \longleftarrow 0$$

where CP is the chain complex coming from the pushout.

In this case the functions $i2, j2, \sigma2$ and $\rho2$ are defined as follows:

$$\begin{aligned}
i2 : CY \oplus CZ &\rightarrow CP \\
y &\mapsto y \quad \text{if } y \in CY \\
z &\mapsto z \quad \text{if } z \in CZ \\
\\
j2 : CP &\rightarrow M \\
x \times \{a\} &\mapsto x \times \{a\} \quad \text{if } x \in X \text{ and } a \in I \\
y &\mapsto nil \quad \text{if } y \in Y \\
z &\mapsto nil \quad \text{if } z \in Z \\
\\
\sigma2 : M &\rightarrow CP \\
x \times \{a\} &\mapsto x \times \{a\} \\
\\
\rho2 : CP &\rightarrow CY \oplus CZ \\
x \times \{a\} &\mapsto nil \quad \text{if } x \in X \text{ and } a \in I \\
y &\mapsto y \quad \text{if } y \in Y \\
z &\mapsto z \quad \text{if } z \in Z
\end{aligned}$$

From $EFHM(Y)$, $EFHM(Z)$ and $EFHM(M)$ we have an algorithm that returns $EFHM(CP)$ from the SES_2 theorem.

The effective short exact sequence generates a connection chain complex morphism $\chi : M_* \rightarrow (CY \oplus CZ)_*^{[1]}$. The “exponent” [1] explains the suspension functor is applied to the chain-complex $(CY \oplus CZ)_*$: the degree of an element is increased by 1 and the differential is replaced by the opposite. The connection morphism is defined as the composition $\chi = \rho d_{CP} \sigma$

Again, we have two homotopy equivalences

$$M \Longleftarrow Cone(\hat{i}) \Longrightarrow Cone(Ei)$$

$$CY \oplus CZ \Longleftarrow \widehat{CY \oplus CZ} \Longrightarrow E(CY \oplus CZ)$$

and the chain complex morphism χ that induces two “parallel” morphisms $\hat{\chi} = (lgyz)\chi(lf'_i)$ and $E\chi = (rfyz)(lgyz)\chi(lf'_i)(rg_i)$:

$$\begin{array}{ccccc}
& \begin{array}{c} lh_{yz} \quad rh_{yz} \\ \curvearrowright \end{array} & & \begin{array}{c} lh'_i \quad rh_i \\ \curvearrowright \end{array} & \\
& CY \oplus CZ_* & \xleftarrow{\hat{\chi}} & Cone(\hat{i})_* & \\
\begin{array}{c} \nearrow lgyz \\ \searrow lfyz \end{array} & & \begin{array}{c} \nearrow rfyz \\ \searrow rgyz \end{array} & & \begin{array}{c} \nearrow lg'_i \\ \searrow lf'_i \end{array} \\
(CY \oplus CZ)_* & & E(CY \oplus CZ)_* & & M_* & & Cone(Ei)_* \\
& \xleftarrow{\chi} & & \xleftarrow{E\chi} & & & \\
& & & & & &
\end{array}$$

The SES_1 theorem said that CP is canonically isomorphic to $Cone(\chi)$, then we have

$$\begin{array}{ccc}
& \begin{array}{c} lh_\chi \quad rh_\chi \\ \curvearrowright \end{array} & \\
& Cone(\hat{\chi})_* & \\
\begin{array}{c} \nearrow lg_\chi \\ \searrow lf_\chi \end{array} & & \begin{array}{c} \nearrow rf_\chi \\ \searrow rg_\chi \end{array} \\
CP \cong Cone(\chi)_* & & Cone(E\chi)_*
\end{array}$$

where:

$$\begin{aligned}
d_{Cone(\chi)_*} &= \begin{bmatrix} d_{(CY \oplus CZ)_*} & \chi \\ 0 & d_{M_*} \end{bmatrix}, \quad d_{Cone(\hat{\chi})_*} = \begin{bmatrix} d_{(\widehat{CY \oplus CZ})_*} & \hat{\chi} \\ 0 & d_{Cone \hat{i}_*} \end{bmatrix}, \\
d_{Cone(E\chi)_*} &= \begin{bmatrix} d_{E(CY \oplus CZ)_*} & E\chi \\ 0 & d_{Cone(Ei)_*} \end{bmatrix}, \quad lf_\chi = \begin{bmatrix} lfy z & 0 \\ 0 & lf'_i \end{bmatrix}, \quad lg_\chi = \begin{bmatrix} lgy z & 0 \\ 0 & lg'_i \end{bmatrix}, \\
lh_\chi &= \begin{bmatrix} lhy z & 0 \\ 0 & -lh'_i \end{bmatrix}, \quad rf_\chi = \begin{bmatrix} rfy z & (rfy z)\hat{\chi}(rh_i) \\ 0 & rf_i \end{bmatrix} \\
rg_\chi &= \begin{bmatrix} rgi & -(rhy z)\hat{\chi}(rg_i) \\ 0 & rg_i \end{bmatrix}, \quad rh_\chi = \begin{bmatrix} rhy z & (rhy z)\hat{\chi}(rh_i) \\ 0 & -rh_i \end{bmatrix}
\end{aligned}$$

In this way from $EFHM(X)$, $EFHM(Y)$ and $EFHM(Z)$ ($EFHM(X \times I)$ can be obtained from $EFHM(X)$ and $EFHM(I)$) we have an equivalence between CP and an effective chain complex.

2 Kenzo Files

In order to build a concrete program to compute $EFHM(CP)$, we are going to follow the steps of the proof for the Theorem 4.

The organization of the next subsections is as follows (this organization is used in the Kenzo manual [1]): first, we give a small introduction of each one of the files and the functions of them. After that, we introduce some examples of the use of the files, more examples can be found in each one of the files. In some of the subsections we introduce the way of searching the homology. Finally, we give the list of Lisp files concerned in each subsection.

2.1 Direct Sum

Let X and Y two spaces, the programs described in this subsection builds $X \oplus Y$

The generators of $X \oplus Y$ are represented internally in the system by a lisp object of the form:

```
(:direct-sum-gsm (d-sum-ind.d-sum-old))
```

where,

1. **d-sum-ind** is a non negative integer with value 0 or 1. 0 indicates the origin of the simplex is X and 1 for Y .
2. **d-sum-old** is a non degenerate simplex, coming from one of the arguments X or Y as indicated by the value of d-sum-ind.

direct-sum-cmpr *cmpr1 cmpr2* [Function]

From the comparison functions *cmpr1* and *cmpr2*, build a comparison function to compare two generators of a direct sum.

direct-sum-basis *basis1 basis2* [Function]

From the functions *basis1* and *basis2* of the chain complexes X and Y , build a basis function for the direct sum of these chain complexes.

direct-sum-cmbn-split *cmbn* [Function]

From a combination of elements of the direct sum of X and Y , it returns two combinations the first one with elements of X and the second one with the elements of Y .

direct-sum-dffr *dffr1 dffr2* [Function]

From the lisp differential functions *dffr1* and *dffr2* of two chain complexes, build the lisp differential function of the direct sum of these chain complexes.

direct-sum *chcm1 chcm2* [Method]

Build the chain complex direct sum of the chain complexes *chcm1* and *chcm2*, using the basic functions above, as shown in the following call to `build-chcm`:

```
(the chain-complex
  (build-chcm
    :cmpr (direct-sum-cmpr (cmpr chcm1) (cmpr chcm2))
    :basis (direct-sum-basis (basis chcm1) (basis chcm2))
    :bsgn (direct-sum-gsm 0 (bsgn chcm1))
    :intr-dffr (direct-sum-dffr (dffr chcm1) (dffr chcm2))
    :strt :cmbn
    :orgn '(direct-sum ,chcm1 ,chcm2))
```

direct-sum-mrph *sorc trgt mrph1 mrph2* [Function]

Build the direct-sum of the morphisms *mrph1* and *mrph2*. This is a morphism of the same degree as *mrph1*. Return the morphism built by the following call to `build-mrph`:

```
(build-mrph
  :sorc sorc
  :trgt trgt
  :degr (degr mrph1)
  :intr (direct-sum-dffr mrph1 mrph2)
  :strt :cmbn
  :orgn '(direct-sum-mrph ,sorc ,trgt ,mrph1 ,mrph2))
```

direct-sum-efhm *chcm1 chcm2* [Function]

Build the homotopy equivalence of the direct sum of *chcm1* and *chcm2*.

Examples

```
(cat-init)
```

```
---done---
```

```

(setf s3 (sphere 3))

[K1 Simplicial-Set]

(setf s3+s3 (direct-sum s3 s3))

[K6 Chain-Complex]

(homology s3+s3 0 5)

Homology in dimension 0 :

Component Z

Component Z

---done---

Homology in dimension 1 :

---done---

Homology in dimension 2 :

---done---

Homology in dimension 3 :

Component Z

Component Z

---done---

Homology in dimension 4 :

---done---

(setf ls3 (loop-space (sphere 3)))

[K17 Simplicial-Group]

(setf ls3+s3 (direct-sum ls3 s3))

[K29 Chain-Complex]

(homology ls3 0 6)

Homology in dimension 0 :

```


Component Z

---done---

Homology in dimension 1 :

---done---

Homology in dimension 2 :

Component Z

---done---

Homology in dimension 3 :

---done---

Homology in dimension 4 :

Component Z

---done---

Homology in dimension 5 :

---done---

(homology ls3+s3 0 6)

Homology in dimension 0 :

Component Z

Component Z

---done---

Homology in dimension 1 :

---done---

Homology in dimension 2 :

Component Z

---done---

Homology in dimension 3 :

Component Z

---done---

Homology in dimension 4 :

Component Z

---done---

Homology in dimension 5 :

---done---

Searching homology for direct sum

The comment list of a direct sum has the form (DIRECT-SUM object1 object2). The `search-efhm` method applied to a wedge, looks for the value of the `efhm` slot of the two objects (i.e. two homotopy equivalences), then builds the direct sum from these homotopy equivalences.

Lisp files concerned in this chapter

direct-sum.cl

2.2 AiBjC reduction

The algorithms of this section are used to build the reduction $A \Longleftarrow Cone(i)$.

$$\begin{array}{ccc} & \xrightarrow{h} & \\ & \swarrow & \\ Cone(i)_* & \xrightleftharpoons[g]{f} & A_* \end{array}$$

These algorithms were extracted from the GiftQ program of Francis Sergeraert.

AiBjC-RDCT-F *A i rho B j sigma C [Function]*

Build the f morphism of the reduction.

AiBjC-RDCT-G *A i rho B j sigma C [Function]*

Build the g morphism of the reduction.

AiBjC-RDCT-H *A i rho B j sigma C [Function]*

Build the h morphism of the reduction.

AiBjC-RDCT *A i rho B j sigma C [Function]*

Build the reduction using the basic functions above, as shown in the following call to `build-rdct`:

```

(build-rdct
 :f (AiBjC-rdct-f A i rho B j sigma C)
 :g (AiBjC-rdct-g A i rho B j sigma C)
 :h (AiBjC-rdct-h A i rho B j sigma C)
 :orgn '(AiBjC-rdct ,A ,i ,rho ,B ,j ,sigma ,C))

```

Lisp files concerned in this chapter

AiBjC-rdct.cl

2.3 Remove Covers

Let X a simplicial set, the programs described in this subsection builds the chain complex M coming from $X \times I$ but with the simplices of $X \times \{0\}$ and $X \times \{1\}$ cancelled.

The generators of the chain complex coming from $X \times I$ are represented internally in the system by a lisp object of the form:

```
(:crpr (dgop1.gmsm1).(dgop2.gmsm2))
```

where,

1. **dgop1** is an integer representing a coded degeneracy operator.
2. **gmsm1** is a non-degenerate simplex of X , to which is applied the degeneracy operator dgop1.
3. **dgop2** is an integer representing a coded degeneracy operator.
4. **gmsm2** is a non-degenerate simplex of I , to which is applied the degeneracy operator dgop2. I can be represented in Kenzo as the standard simplex of dimension 1, this Simplicial Set has two vertices represented as 1 and 2 and an edge 3.

The generators of the chain complex coming from $X \times I$ are represented internally in the system by a lisp object of the form:

```
(:crpr (dgop1.gmsm1).(dgop2.gmsm2))
```

where,

1. **dgop1** is an integer representing a coded degeneracy operator.
2. **gmsm1** is a non-degenerate simplex of X , to which is applied the degeneracy operator dgop1.
3. **dgop2** is an integer representing a coded degeneracy operator.
4. **gmsm2** is the non-degenerate simplex 3 of I , because the simplices of $X \times \{0\}$ and $X \times \{1\}$ are cancelled.

remove-covers-basis *basis1* [Function]

From the functions *basis1* of the simplicial set X , build a basis function for the chain complex from $X \times I$ but with the simplices of $X \times \{0\}$ and $X \times \{1\}$ cancelled.

direct-sum-dffr *dffr1* [Function]

From the lisp differential functions *dffr1* of a simplicial set, build the lisp differential function of the the chain complex from $X \times I$ but with the simplices of $X \times \{0\}$ and $X \times \{1\}$ cancelled.

remove-covers *smst1* [Function]

Build the chain complex from $X \times I$ but with the simplices of $X \times \{0\}$ and $X \times \{1\}$ cancelled, using the basic functions above and the comparison function of $X \times I$, as shown in the following code:

```
(defun remove-covers (smst1)
  (declare (type simplicial-set smst1))
  (let ((XxI (crt-prdc smst1 (delta 1))))
    (the chain-complex
      (build-chcm
        :cmpr (cmpr XxI)
        :basis (remove-covers-basis (basis XxI))
        :bsgn nil
        :intr-dffr (remove-covers-dffr (dffr XxI))
        :strt :cmbn
        :orgn '(remove-covers ,smst1))))))
```

remove-covers-efhm *smst1* [Function]

Build the homotopy equivalence of the chain complex from $X \times I$ but with the simplices of $X \times \{0\}$ and $X \times \{1\}$ cancelled, using the functions from the AiBjC-rdct file.

Examples

```
(cat-init)

---done---

(setf s3 (sphere 3))

[K1 Simplicial-Set]

(setf rcs3 (remove-covers s3))

[K16 Chain-Complex]

(homology rcs3 0 6)

Homology in dimension 0 :

---done---

Homology in dimension 1 :

Component Z
```

```

---done---

Homology in dimension 2 :

---done---

Homology in dimension 3 :

---done---

Homology in dimension 4 :

Component Z

Homology in dimension 5 :

---done---

(setf ls3 (loop-space (sphere 3)))

[K99 Simplicial-Group]

(setf rcls3 (remove-covers ls3))

[K116 Chain-Complex]

(homology rcls3 0 6)

Homology in dimension 0 :

---done---

Homology in dimension 1 :

Component Z

---done---

Homology in dimension 2 :

---done---

Homology in dimension 3 :

Component Z

---done---

Homology in dimension 4 :

```

---done---

Homology in dimension 5 :

Component Z

---done---

Lisp files concerned in this chapter

remove-covers.cl, AiBjC-rdct.cl

Searching homology for remove-covers

The comment list of a direct sum has the form (`remove-covers object1`). The `search-efhm` method applied to a remove-covers, looks for the value of the `efhm` slot of the object (i.e. a homotopy equivalences), using the functions from AiBjC-rdct.lisp builds the homotopy equivalence for the chain complex from $X \times I$ but with the simplices of $X \times \{0\}$ and $X \times \{1\}$ cancelled.

2.4 Suspension

In Kenzo the suspension process is realized by the function `suspension`. This function has one argument, a reduced object, but if the object is not reduced, the result is undefined. Then, we have undertaken the task of developing the functor `suspension`.

`suspension2-basis basis` [Function]

From the functions *basis* of a chain complex, build a basis function for the suspension of this chain complex.

`suspension2-dffr dffr` [Function]

From the lisp differential functions *dffr* of a chain complex, build the lisp differential function for the suspension of this chain complex.

`suspension2 chcm` [Method]

Build the chain complex suspension of the chain complex *chcm*, using the basic functions above, as shown in the following call to `build-chcm`:

```
(build-chcm
  :cmpr cmpr
  :basis (suspension2-basis basis)
  :bsgn nil
  :intr-dffr (suspension2-intr-dffr dffr)
  :strt :cmbn
  :orgn '(suspension2 ,chcm))
```

`suspension2-intr mrph` [Function]

From a Kenzo morphism *mrph*, build an internal function corresponding to the suspension of the initial morphism.

suspension2 *mrph* [Method]

Build the suspension of the morphism *mrph*.

suspension2 *rdct* [Method]

Build the suspension of the reduction *rdct*.

suspension2 *hmeq* [Method]

Build the suspension of the homotopy equivalence *hmeq*.

Examples

```
(cat-init)
```

```
---done---
```

```
(setf k-z (k-z 1))
```

```
[K1 Abelian-Simplicial-Group]
```

```
(setf sk-z (suspension2 k-z))
```

```
[K13 Chain-Complex]
```

```
(homology sk-z 0 6)
```

```
Homology in dimension 0 :
```

```
---done---
```

```
Homology in dimension 1 :
```

```
Component Z
```

```
---done---
```

```
Homology in dimension 2 :
```

```
Component Z
```

```
---done---
```

```
Homology in dimension 3 :
```

```
---done---
```

```
Homology in dimension 4 :
```

```
Homology in dimension 5 :
```

```
---done---
```

Lisp files concerned in this chapter

suspension2.cl

2.5 Cones

The cones are defined in the Kenzo system, but with the definition: Let C_* and D_* be two chain-complexes and $\phi : C_* \leftarrow D_*$ be a chain-complex morphism. Then the cone of ϕ denoted by $Cone(\phi)$ is the chain complex $Cone(\phi) = A_*$ defined as follows. First $A_n := C_n \oplus D_{n-1} \dots$

But, the SES_2 theorem works with the alternative definition: Let C_* and D_* be two chain-complexes and $\phi : C_* \leftarrow D_*$ be a chain-complex morphism. Then the cone of ϕ denoted by $Cone(\phi)$ is the chain complex $Cone(\phi) = A_*$ defined as follows. First $A_n := C_{n+1} \oplus D_n \dots$

Then, we have defined a new file for cones based on the previous one.

2.5.1 Representation of a combination in a cone

To distinguish to which chain complex belongs a generator in a combination of a cone, the following convention has been adopted: if **gc** is a generator of any degree of C , it will be represented in the cone by the list `(:con 0 gc)` and printed as `<CONE 0 gc>`. The symbol for D is `(:con 1 gd)`.

2.5.2 Useful functions and macros for cones

con0 gnrt [Macro]

Build the representation of the generator *gnrt* belonging to the chain complex C .

con1 gnrt [Macro]

Build the representation of the generator *gnrt* belonging to the chain complex D .

cone-cmbn-split2 cmbn [Function]

Give the 2-values result constituted by the 2 combinations components of the cone combination *cmbn*. These combination are valid combinations in their respective chain complexes. If *cmbn* has degree n then the combination of C has degree $n + 1$ and the combination of D has degree n .

2.6 Construction of a cone from a morphism

From the slots of the morphism $\phi : C_* \leftarrow D_*$, it is possible to build the cone chain complex. The three essential functions are the following:

cone-cmpr cmpr0 cmpr1 [Function]

From the 2 comparison functions *cmpr0* and *cmpr1*, build a comparison function adequate to compare the generators as represented in the cone

cone2-basis *basis0 basis1* [Function]

From the 2 basis function *basis0* and *basis1*, build a basis function for the cone. If at least one of the chain complex component of the cone is *locally effective*, the function returns the symbol **:locally-effective**

cone-3mrph-triangle-impl2 *cmpr0 mrph0 mrph1 phi* [Function]

Define the differential in the cone according to the formula:

$$d(cc, cd) = (d_C(cc), \phi(cc) - d_D(cd))$$

cone2 mrph [Method]

Build Cone(*mrph*), using the above functions.

cone2-efhm mrph [Function]

From the formulas of the Cone Equivalence Theorem we can build the homotopy equivalence for the Cone(*mrph*).

Examples

```
(cat-init)
```

```
---done---
```

```
(setf k (k-z 1))
```

```
[K1 Abelian-Simplicial-Group]
```

```
(setf u (idnt-mrph k))
```

```
[K13 Morphism (degree 0): K1 -> K1]
```

```
(setf c (cone2 u))
```

```
[K15 Chain-Complex]
```

```
(setf *tc* (cmbn 3 1 (con0 '(1 2 3 4)) 2 (con1 '(1 2 3 4))))
```

```
-----{CMBN 3}  
<1 * <CONE 0 (1 2 3 4)>>  
<2 * <CONE 1 (1 2 3 4)>>  
-----
```

```
(? c *tc*)
```

```
-----{CMBN 2}  
<2 * <CONE 0 (1 2 3 4)>>  
<1 * <CONE 0 (1 2 3)>>  
<-1 * <CONE 0 (1 2 7)>>  
<1 * <CONE 0 (1 5 4)>>
```

```

<1 * <CONE 0 (2 3 4)>>
<-1 * <CONE 0 (3 3 4)>>
<2 * <CONE 1 (1 2 3)>>
<-2 * <CONE 1 (1 5 4)>>
<-2 * <CONE 1 (2 3 4)>>
<2 * <CONE 1 (3 3 4)>>

```

Lisp files concerned in this chapter

cones2.cl, cones.cl

3 Pushout

This is the main file and is able to build the pushout from $f : X \rightarrow Y$ and $g : X \rightarrow Z$ simplicial morphisms, we will denote the pushout as (f, g) .

The *non-degenerate* simplex of (f, g) are represented internally in the system by a lisp object of the form:

(`:pushout-gsm (p-ind.p-old)`)

where,

1. **p-ind** is a non negative integer with value 0, 1 or 2. p-ind indicates the origin of the simplex, 0 for $X \times I$, 1 for Y and 2 for Z .
2. **p-old** is a non degenerate simplex of dimension, coming from one of the arguments $X \times I$ or Y or Z as indicated by the value of p-ind.

pushout-cmpr *cmpr1 cmpr2 cmpr3* [Function]

From the comparison functions *cmpr1*, *cmpr2* and *cmpr3*, build a comparison function to compare two generators of a pushout.

pushout-basis *basis1 basis2 basis3* [Function]

From the functions *basis1*, *basis2* and *basis3* of the simplicial sets $X \times I$, Y and Z , build a basis function for the pushout of these simplicial sets.

pushout-face *face1 face2 face3 f g* [Function]

From the face operators *face1*, *face2* and *face3* and the morphism f and g , builds the face operator for the pushout.

pushout f g [Method]

Build the simplicial set pushout of the simplicial morphisms f and g , using the basic functions above, as shown in the following call to **build-smst**:

```

(let* ((X (sorc f))
      (XxI (crts-prdc X (delta 1)))
      (Y (trgt f))
      (Z (trgt g)))
  (the simplicial-set

```

```

(let ((rslt (build-smst
               :cmpr (pushout-cmpr (cmpr XxI) (cmpr Y) (cmpr Z))
               :basis (pushout-basis (basis XxI) (basis Y) (basis Z))
               :bspn (pushout-gsm 1 (bspn Y))
               :face (pushout-face (face XxI) (face Y) (face Z) f g)
               :orgn '(pushout ,f ,g))))
      (declare (type simplicial-set rslt))
      rslt)))

```

pushout-efhm *f g* [*Function*]

Applying the process explained in Section 1, this function returns the homotopy equivalence for the pushout.

Examples

From the pushout, we can generate several constructions.

Wedge

Let Y and Z two reduced simplicial sets and X the simplicial set which have only 1 vertex. f and g are the morphism that goes from the vertex of X to the base point of Y and Z respectively.

```

(cat-init)

---done---

(setf bk-z (k-z 2))

[K13 Abelian-Simplicial-Group]

(setf unipunctual (build-finite-ss '(x)))

Checking the 0-simplices...
[K25 Simplicial-Set]

(setf f (build-smmr :sorc unipunctual :trgt bk-z :degr 0
                   :sintr #'(lambda (dmns gmsm)
                               (declare (ignore gmsm))
                               (if (= dmns 0)
                                   (absm 0 (bspn bk-z))
                                   nil)
                               ))
         :orgn '(triv ,unipunctual ,bk-z)))

[K30 Simplicial-Morphism K25 -> K13]

(setf p (pushout f f))

[K41 Simplicial-Set]

```

```

(homology p 0 10)

Homology in dimension 0 :

Component Z

---done---

Homology in dimension 1 :

---done---

Homology in dimension 2 :

Component Z

---done---

Homology in dimension 3 :

---done---

Homology in dimension 4 :

Component Z

Component Z

---done---

Homology in dimension 5 :

---done---

Homology in dimension 6 :

Component Z

Component Z

---done---

Homology in dimension 7 :

---done---

Homology in dimension 8 :

Component Z

```

Component Z

---done---

Homology in dimension 9 :

---done---

Join

If Y and Z are arbitrary spaces, then the pushout of both projections $f : Y \times Z \rightarrow Y$ and $g : Y \times Z \rightarrow Z$ is nothing but the *join* of Y and Z .

Example: The join of the spheres S^n and S^m is the sphere S^{n+m+1} .

(cat-init)

---done---

(setf s3 (sphere 3))

[K1 Simplicial-Set]

(setf s4 (sphere 4))

[K6 Simplicial-Set]

(setf s3xs4 (crts-prdc s3 s4))

[K11 Simplicial-Set]

```
(setf f (build-smmr :sorc s3xs4 :trgt s3 :degr 0
                   :sintr #'(lambda (dmns gmsm)
                               (declare (ignore dmns))
                               (absm (dgop1 gmsm) (gmsm1 gmsm))))
      :orgn '(projection ,s3xs4 ,s3)))
```

[K16 Simplicial-Morphism K11 -> K1]

```
(setf g (build-smmr :sorc s3xs4 :trgt s4 :degr 0
                   :sintr #'(lambda (dmns gmsm)
                               (declare (ignore dmns))
                               (absm (dgop2 gmsm) (gmsm2 gmsm))))
      :orgn '(projection ,s3xs4 ,s4)))
```

[K17 Simplicial-Morphism K11 -> K1]

(setf p (pushout f g))

[K28 Simplicial-Set]

(homology p 0 9)

Homology in dimension 0 :

Component Z

---done---

Homology in dimension 1 :

---done---

Homology in dimension 2 :

---done---

Homology in dimension 3 :

---done---

Homology in dimension 4 :

---done---

Homology in dimension 5 :

---done---

Homology in dimension 6 :

---done---

Homology in dimension 7 :

---done---

Homology in dimension 8 :

Component Z

---done---

Homology in dimension 9 :

---done---

Direct Sum

If $X = \emptyset$, then the pushout is simply the disjoint union of Y and Z .

```

(cat-init)

---done---

(setf l2s3 (loop-space (sphere 3) 2))

[K18 Simplicial-Group]

(setf l3s4 (loop-space (sphere 4) 3))

[K59 Simplicial-Group]

(setf nil-ss (build-finite-ss nil))

Checking the 0-simplices...
[K71 Simplicial-Set]

(setf f (build-smmr :sorc nil-ss :trgt l2s3 :degr 0
                   :sintr #'(lambda (dmns gmsm)
                               nil)
                   :orgn '(triv ,nil-ss ,l2s3)))

[K76 Simplicial-Morphism K71 -> K18]

(setf g (build-smmr :sorc nil-ss :trgt l3s4 :degr 0
                   :sintr #'(lambda (dmns gmsm)
                               nil)
                   :orgn '(triv ,nil-ss ,l3s4)))

[K77 Simplicial-Morphism K71 -> K59]

(setf p (pushout f g))

[K88 Simplicial-Set]

(setf ds (direct-sum l2s3 l3s4))

[K93 Chain-Complex]

(homology p 0 6)

Homology in dimension 0 :

Component Z

Component Z

---done---

```

Homology in dimension 1 :

Component \mathbb{Z}

Component \mathbb{Z}

---done---

Homology in dimension 2 :

Component $\mathbb{Z}/2\mathbb{Z}$

Component $\mathbb{Z}/2\mathbb{Z}$

---done---

Homology in dimension 3 :

Component $\mathbb{Z}/2\mathbb{Z}$

Component $\mathbb{Z}/2\mathbb{Z}$

---done---

Homology in dimension 4 :

Component $\mathbb{Z}/3\mathbb{Z}$

Component $\mathbb{Z}/3\mathbb{Z}$

Component $\mathbb{Z}/2\mathbb{Z}$

Component $\mathbb{Z}/2\mathbb{Z}$

Component \mathbb{Z}

---done---

Homology in dimension 5 :

Component $\mathbb{Z}/3\mathbb{Z}$

Component $\mathbb{Z}/3\mathbb{Z}$

Component $\mathbb{Z}/2\mathbb{Z}$

Component $\mathbb{Z}/2\mathbb{Z}$

Component \mathbb{Z}

---done---

(homology ds 0 6)

Homology in dimension 0 :

Component \mathbb{Z}

Component \mathbb{Z}

---done---

Homology in dimension 1 :

Component \mathbb{Z}

Component \mathbb{Z}

---done---

Homology in dimension 2 :

Component $\mathbb{Z}/2\mathbb{Z}$

Component $\mathbb{Z}/2\mathbb{Z}$

---done---

Homology in dimension 3 :

Component $\mathbb{Z}/2\mathbb{Z}$

Component $\mathbb{Z}/2\mathbb{Z}$

---done---

Homology in dimension 4 :

Component $\mathbb{Z}/3\mathbb{Z}$

Component $\mathbb{Z}/3\mathbb{Z}$

Component $\mathbb{Z}/2\mathbb{Z}$

Component $\mathbb{Z}/2\mathbb{Z}$

Component \mathbb{Z}

---done---

Homology in dimension 5 :

Component $\mathbb{Z}/3\mathbb{Z}$

Component $\mathbb{Z}/3\mathbb{Z}$

Component $\mathbb{Z}/2\mathbb{Z}$

Component $\mathbb{Z}/2\mathbb{Z}$

Component \mathbb{Z}

---done---

$P^2(\mathbb{C})$

A sophisticated example giving a geometrical construction of $P^2(\mathbb{C})$. You take S^2 and you construct the first stage of the Whitehead tower by doing:

(cat-init)

---done---

(setf s2 (sphere 2))

[K1 Simplicial-Set]

(setf ch2 (chml-class s2 2))

[K12 Cohomology-Class on K1 of degree 2]

(setf f2 (z-whitehead s2 ch2))

[K25 Fibration K1 -> K13]

(setf x3 (fibration-total f2))

[K31 Simplicial-Set]

(setf unipunctual (build-finite-ss '(x)))

[K36 Simplicial-Set]

Then x_3 has the homotopy type of the 3-sphere S^3 . More precisely $x_3 = s_2 \times_{f_2} K(\mathbb{Z}, 1)$ with f_2 an appropriate twisting function producing S^3 as a total space.

It is easy to deduce a projection $f : X_3 \rightarrow S^2$. If you take the pushout of this f and the only map $g : X_3 \rightarrow *$, then the pushout is $P^2(\mathbb{C})$, which

homology groups are $H_n(P^2(\mathbb{C})) = \mathbb{Z}$ if $n = 0, 2, 4$ and $H_n(P^2(\mathbb{C})) = 0$ in the rest of cases as we can see:

```
(setf f (build-smmr :sorc x3 :trgt s2 :degr 0
                   :sintr #'(lambda (dmns gmsm)
                               (declare (ignore dmns))
                               (absm (dgop1 gmsm) (gmsm1 gmsm))))
  :orgn '(proj ,x3 ,s2)))
```

[K41 Simplicial-Morphism K31 -> K1]

```
(setf g (build-smmr :sorc x3 :trgt unipunctual :degr 0
                   :sintr #'(lambda (dmns gmsm)
                               (if (and (equal dmns 0)
                                           (equal gmsm (bsgn x3)))
                                   'x
                                   nil)))
  :orgn '(proj ,x3 ,unipunctual)))
```

[K42 Simplicial-Morphism K31 -> K36]

```
(setf p (pushout f g))
```

[K53 Simplicial-Set]

```
(homology p 0 10)
```

Homology in dimension 0 :

Component Z

---done---

Homology in dimension 1 :

---done---

Homology in dimension 2 :

Component Z

---done---

Homology in dimension 3 :

---done---

Homology in dimension 4 :

Component Z

---done---

Homology in dimension 5 :

---done---

Homology in dimension 6 :

---done---

Homology in dimension 7 :

---done---

Homology in dimension 8 :

---done---

Homology in dimension 9 :

---done---

Lisp files concerned in this chapter

pushout.cl, [cones2.cl], [remove-covers.cl], [suspension2.cl], [direct-sum.cl], [AiBjC-rdct.cl].

References

- [1] Dousson X., Sergeraert F., Siret Y., *The Kenzo program*, Institut Fourier, Grenoble, 1999.
<http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo/>.
- [2] Mather M., *Pull-backs in Homotopy Theory*, Can. J. Math. 28-2 (1976), 225–263.
- [3] Rubio J., Sergeraert F. *Constructive Homological Algebra and Applications*, Genova Summer School 2006.
<http://www-fourier.ujf-grenoble.fr/~sergerar/Papers/Genova-Lecture-Notes.pdf>