Chapter 1

Preliminaries

In this chapter we introduce some preliminaries that we will use in the rest of this memoir. The first section is devoted to the main mathematical notions employed in this work. The Kenzo system, a Common Lisp program developed by Francis Sergeraert and some coworkers devoted to perform computations in Algebraic Topology, is presented in Section 1.2. Finally, the deduction machinery employed in this memoir, the ACL2 system, is briefly introduced in Section 1.3.

1.1 Mathematical preliminaries

Algebraic Topology is a vast and complex subject, in particular mixing Algebra and (combinatorial) Topology. Algebraic Topology consists in trying to use as much as possible “algebraic” methods to attack topological problems. For instance, one can define some special groups associated with a topological space, in a way that respects the relation of homeomorphism of spaces. This allows us to study properties about topological spaces by means of statements about groups, which are often easier to prove.

1.1.1 Homological Algebra

The following basic definitions can be found, for instance, in [Mac63].

**Definition 1.1.** Let \( R \) be a ring with a unit element \( 1 \neq 0 \). A left \( R \)-module \( M \) is an additive abelian group together with a map \( p : R \times M \to M \), denoted by \( p(r, m) \equiv rm \), such that for every \( r, r' \in R \) and \( m, m' \in M \)

\[
\begin{align*}
(r + r')m &= rm + r'm \\
r(m + m') &= rm + rm' \\
(rr')m &= r(r'm) \\
1m &= m
\end{align*}
\]
A similar definition is given for a \textit{right} \( R \)-module.

For \( R = \mathbb{Z} \) (the integer ring), a \( \mathbb{Z} \)-module \( M \) is simply an abelian group. The map \( p : \mathbb{Z} \times M \to M \) is given by

\[
p(n, m) = \begin{cases} 
  m + \cdots + m & \text{if } n > 0 \\
  0 & \text{if } n = 0 \\
  (-m) + \cdots + (-m) & \text{if } n < 0
\end{cases}
\]

\textbf{Definition 1.2.} Let \( R \) be a ring and \( M \) and \( N \) be \( R \)-modules. An \( R \)-module morphism \( \alpha : M \to N \) is a function from \( M \) to \( N \) such that for every \( m, m' \in M \) and \( r \in R \)

\[
\alpha(m + m') = \alpha(m) + \alpha(m') \quad \alpha(rm) = r\alpha(m) \quad \alpha(0_M) = 0_N
\]

\textbf{Definition 1.3.} Given a ring \( R \), a \textit{graded module} \( M \) is a family of left \( R \)-modules \( (M_n)_{n \in \mathbb{Z}} \).

\textbf{Definition 1.4.} Given a pair of graded modules \( M \) and \( M' \), a \textit{graded module morphism} \( f \) of degree \( k \) between them is a family of module morphisms \( (f_n)_{n \in \mathbb{Z}} \) such that \( f_n : M_n \to M'_{n+k} \) for all \( n \in \mathbb{Z} \).

\textbf{Definition 1.5.} Given a graded module \( M \), a \textit{differential} \( (d_n)_{n \in \mathbb{Z}} \) is a family of module endomorphisms of \( M \) of degree \(-1\) such that \( d_{n-1} \circ d_n = 0 \) for all \( n \in \mathbb{Z} \).

From the previous definitions, the notion of chain complex can be defined. Chain complexes are the central notion in Homological Algebra and can be used as an algebraic means to study properties of topological spaces in several dimensions.

\textbf{Definition 1.6.} A \textit{chain complex} \( C_* \) is a family of pairs \( (C_n, d_n)_{n \in \mathbb{Z}} \) where \( (C_n)_{n \in \mathbb{Z}} \) is a graded module and \( (d_n)_{n \in \mathbb{Z}} \) is a differential of \( C_* \).

The module \( C_n \) is called the module of \textit{n-chains}. The image \( B_n = \text{Im} d_{n+1} \subseteq C_n \) is the (sub)module of \textit{n-boundaries}. The kernel \( Z_n = \text{Ker} d_n \subseteq C_n \) is the (sub)module of \textit{n-cycles}.

In many situations the ring \( R \) is the integer ring, \( R = \mathbb{Z} \). In this case, a chain complex \( C_* \) is given by a graded abelian group \( (C_n)_{n \in \mathbb{Z}} \) and a graded group morphism of degree \(-1\), \( \{d_n : C_n \to C_{n-1}\}_{n \in \mathbb{Z}} \), satisfying \( d_{n-1} \circ d_n = 0 \) for all \( n \). From now on in this memoir, we will work with \( R = \mathbb{Z} \).

Let us present some examples of chain complexes.

\textbf{Example 1.7.} \begin{itemize}
  
  \item The \textit{unit chain complex} has a unique non null component, namely a \( \mathbb{Z} \)-module in degree 0 generated by a unique generator and the differential is the null map.
\end{itemize}
• A chain complex to model the circle is defined as follows. This chain complex has two non null components, namely a $\mathbb{Z}$-module in degree 0 generated by a unique generator and a $\mathbb{Z}$-module in degree 1 generated by another generator; and the differential is the null map.

• The diabolo chain complex has associated three chain groups:
  
  - $C_0$, the free $\mathbb{Z}$-module on the set $\{s_0, s_1, s_2, s_3, s_4, s_5\}$.
  - $C_1$, the free $\mathbb{Z}$-module on the set $\{s_{01}, s_{02}, s_{12}, s_{23}, s_{34}, s_{45}, s_{35}\}$.
  - $C_2$, the free $\mathbb{Z}$-module on the set $\{s_{345}\}$.

  and the differential is provided by:
  
  - $d_0(s_i) = 0$,
  - $d_1(s_{ij}) = s_j - s_i$,
  - $d_2(s_{ijk}) = s_{jk} - s_{ik} + s_{ij}$.

  and it is extended by linearity to the combinations $c = \sum_{i=1}^{m} \lambda_i x_i \in C_n$.

We can construct chain complexes from other ones, applying constructors such as the direct sum or the tensor product.

**Definition 1.8.** Let $C_* = (C_n, d_{C_n})_{n \in \mathbb{Z}}$ and $D_* = (D_n, d_{D_n})_{n \in \mathbb{Z}}$ be chain complexes. The **direct sum** of $C_*$ and $D_*$ is the chain complex $C_* \oplus D_* = (M_n, d_n)_{n \in \mathbb{Z}}$ such that, $M_n = (C_n, D_n)$ and the differential map is defined on the generators $(x, y)$ with $x \in C_n$ and $y \in D_n$ by $d_n((x, y)) = (d_{C_n}(x), d_{D_n}(y))$ for all $n \in \mathbb{Z}$. The notion of direct sum can be generalized to a collection of chain complexes.

**Definition 1.9.** Let $M$ be a right $R$-module, and $N$ a left $R$-module. The **tensor product** $M \otimes_R N$ is the abelian group generated by the symbols $m \otimes n$ for every $m \in M$ and $n \in N$, subject to the relations

$$
(m + m') \otimes n = m \otimes n + m' \otimes n
$$

$$
m \otimes (n + n') = m \otimes n + m \otimes n'
$$

$$
rm \otimes n = m \otimes rn
$$

for all $r \in R$, $m, m' \in M$, and $n, n' \in N$.

If $R = \mathbb{Z}$ (the integer ring), then $M$ and $N$ are abelian groups and their tensor product will be denoted simply by $M \otimes N$.

**Definition 1.10.** Let $C_* = (C_n, d_{C_n})_{n \in \mathbb{Z}}$ and $D_* = (D_n, d_{D_n})_{n \in \mathbb{Z}}$ be chain complexes of right and left $\mathbb{Z}$-modules respectively. The **tensor product** $C_* \otimes D_*$ is the chain complex of $\mathbb{Z}$-modules $C_* \otimes D_* = ((C_* \otimes D_*)_n, d_n)_{n \in \mathbb{Z}}$ with

$$(C_* \otimes D_*)_n = \bigoplus_{p+q=n} (C_p \otimes D_q)$$
where the differential map is defined on the generators $x \otimes y$ with $x \in C_p$ and $y \in D_q$, according to the Koszul rule for the signs, by

$$d_n(x \otimes y) = d_{C_p}(x) \otimes y + (-1)^p x \otimes d_{D_q}(y)$$

Let us present now, one of the most important invariants used in Homological Algebra. Given a chain complex $C^* = (C^n, d_n)_{n \in \mathbb{Z}}$, the identities $d_n - 1 \circ d_n = 0$ mean the inclusion relations $B_n \subseteq Z_n$: every boundary is a cycle (the converse in general is not true). Thus the next definition makes sense.

**Definition 1.11.** Let $C^* = (C^n, d_n)_{n \in \mathbb{Z}}$ be a chain complex of $R$-modules. For each degree $n \in \mathbb{Z}$, the $n$-homology module of $C^*$ is defined as the quotient module

$$H_n(C^*) = \frac{Z_n}{B_n}$$

It is worth noting that the homology groups of a space $X$ are the ones of its associated chain complex $C_*(X)$; the way of constructing the chain complex associated with a space $X$ is explained, for instance, in [Mau96]. In an intuitive sense, homology groups measure “$n$-dimensional holes” in topological spaces. $H_0$ measures the number of connected components of a space. The homology groups $H_n$ measure higher dimensional connectedness. For instance, the $n$-sphere, $S^n$, has exactly one $n$-dimensional hole and no $m$-dimensional holes if $m \neq n$.

Moreover, it is worth noting that homology groups are an invariant, see [Mau96]. That is to say, if two topological spaces are homeomorphic, this means that their homology groups are isomorphic.

Let us finish this section with some additional definitions related to chain complexes.

**Definition 1.12.** A chain complex $C^* = (C_n, d_n)_{n \in \mathbb{Z}}$ is acyclic if $H_n(C^*) = 0$ for all $n$, that is to say, if $Z_n = B_n$ for every $n \in \mathbb{Z}$.

**Definition 1.13.** Let $C^* = (C_n, d_n)_{n \in \mathbb{Z}}$ and $D^* = (D_n, d'_n)_{n \in \mathbb{Z}}$ be two chain complexes, a chain complex morphism between them is a family of module morphisms $(f_n)_{n \in \mathbb{Z}}$ of degree 0 between $(C_n)_{n \in \mathbb{Z}}$ and $(D_n)_{n \in \mathbb{Z}}$ such that $d'_n \circ f_n = f_{n-1} \circ d_n$ for each $n \in \mathbb{Z}$.

**Definition 1.14.** Let $C^* = (C_n, d_n)_{n \in \mathbb{Z}}$ be a chain complex. A chain complex $D^* = (D_n, d'_n)_{n \in \mathbb{Z}}$ is a chain subcomplex of $C^*$ if

- $D_n$ is a submodule of $C_n$, for all $n \in \mathbb{Z}$
- $d'_n = d_n \mid_{D_n}$

The condition $d'_n = d_n \mid_{D_n}$ means that the boundary operator of the chain subcomplex is just the differential operator of the larger chain complex restricted to its domain. We denote $D_s \subset C_s$ if $D_s$ is a chain subcomplex of $C_s$. 
1.1 Mathematical preliminaries

Definition 1.15. A short exact sequence is a sequence of modules:
\[ 0 \leftarrow C'' \xrightarrow{i} C \xrightarrow{j} C' \leftarrow 0 \]
which is exact, that is, the map \( i \) is injective, the map \( j \) is surjective and \( \text{Im } i = \text{Ker } j \).

From now on in this memoir, we will work with non-negative chain complexes, that is to say, \( \{C_n\}_{n \in \mathbb{Z}} \) such that \( C_n = 0 \) if \( n < 0 \). A non-negative chain complex \( C_* \) will be denoted by \( C_* = \{C_n\}_{n \in \mathbb{N}} \). Moreover, the chain complexes we work with are supposed to be free.

Definition 1.16. A chain complex \( C_* = (C_n, d_n)_{n \in \mathbb{N}} \) of \( \mathbb{Z} \)-modules is said to be free if \( C_n \) is a free \( \mathbb{Z} \)-module (a \( \mathbb{Z} \)-module which admits a basis) for each \( n \in \mathbb{N} \).

1.1.2 Simplicial Topology

1.1.2.1 Simplicial Sets

Simplicial sets were first introduced by Eilenberg and Zilber [EZ50], who called them semi-simplicial complexes. They can be used to express some topological properties of spaces by means of combinatorial notions. A good reference for the definitions and results of this section is [May67].

Definition 1.17. A simplicial set \( K \), is a union \( K = \bigcup_{q \geq 0} K^q \), where the \( K^q \) are disjoints sets, together with functions:
\[ \partial^q_i : K^q \rightarrow K^{q-1}, \quad q > 0, \quad i = 0, \ldots, q, \]
\[ \eta^q_i : K^q \rightarrow K^{q+1}, \quad q \geq 0, \quad i = 0, \ldots, q, \]
subject to the relations:
\begin{align*}
(1) \quad & \partial^q_{i+1} \partial^q_{j} = \partial^q_{j} \partial^q_{i} \quad \text{if } i < j, \\
(2) \quad & \eta^q_{i+1} \eta^q_{j} = \eta^q_{j} \eta^q_{i} \quad \text{if } i > j, \\
(3) \quad & \partial^q_{i+1} \eta^q_{j} = \eta^q_{j} \partial^q_{i} \quad \text{if } i < j, \\
(4) \quad & \partial^q_{i+1} \eta^q_{i} = \text{identity} = \partial^q_{i} \eta^q_{i+1}, \\
(5) \quad & \partial^q_{i+1} \eta^q_{j} = \eta^q_{j} \partial^q_{i-1} \quad \text{if } i > j + 1.
\end{align*}

The \( \partial^q_i \) and \( \eta^q_i \) are called face and degeneracy operators respectively.

The elements of \( K^q \) are called \( q \)-simplices. A simplex \( x \) is called degenerate if \( x = \eta_i y \) for some simplex \( y \) and some degeneracy operator \( \eta_i \); otherwise \( x \) is called non degenerate.

An example of a simplicial set, that can be useful to clarify some notions, is the standard simplicial set of dimension \( m \), \( \Delta[m] \).
Chapter 1  Preliminaries

Figure 1.1: non degenerate simplexes of the standard simplicial set $\Delta[3]$

**Definition 1.18.** For $m \geq 0$, the *standard simplicial set of dimension* $m$, $\Delta[m]$, is a simplicial set built as follows. An $n$-simplex of $\Delta[m]$ is any $(n+1)$-tuple $(a_0, \ldots, a_n)$ of integers such that $0 \leq a_0 \leq \cdots \leq a_n \leq m$, and the face and degeneracy operators are defined as

$$
\partial_i(a_0, \ldots, a_n) = (a_0, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n)
$$

$$
\eta_i(a_0, \ldots, a_n) = (a_0, \ldots, a_{i-1}, a_i, a_i, a_{i+1}, \ldots, a_n)
$$

In Definition 1.18 the super-indexes in the degeneracy and face maps have been skipped, since they can be inferred from the context. It is a usual practice and will be freely used in the sequel, both for degeneracy and for face maps.

**Example 1.19.** Figure 1.19 shows the non degenerate simplexes of the standard simplicial set $\Delta[3]$:  

- 0-simplexes (vertices): (0), (1), (2), (3);
- non degenerate 1-simplexes (edges): (0 1), (0 2), (0 3), (1 2), (1 3), (2 3);
- non degenerate 2-simplexes (triangles): (0 1 2), (0 1 3), (0 2 3), (1 2 3); and
- non degenerate 3-simplex (filled tetrahedra): (0 1 2 3).

It is worth noting that there is not any non degenerate $n$-simplex with $n > 3$.

Once we have presented the non degenerate simplexes, let us introduce the behavior of the face and degeneracy maps. For instance, if we apply the face and degeneracy maps over the 2-simplex $(0 1 2)$ (for the rest of simplexes is analogous) we will obtain:

$$
\partial_i((0 1 2)) = \begin{cases} 
(1 2) & \text{if } i = 0 \\
(0 2) & \text{if } i = 1 \\
(0 1) & \text{if } i = 2 
\end{cases}
$$

$$
\eta_i((0 1 2)) = \begin{cases} 
(0 0 1 2) & \text{if } i = 0 \\
(0 1 1 2) & \text{if } i = 1 \\
(0 1 2 2) & \text{if } i = 2 
\end{cases}
$$
1.1 Mathematical preliminaries

Let us note that the face operator applied over the 2-simplex \((0 \ 1 \ 2)\) produces simplexes with geometrical meaning (that are the three edges of the simplex \((0 \ 1 \ 2)\)). On the contrary, the simplexes obtained from applying the degeneracy operator do not have any geometrical meaning.

In the rest of the memoir a non degenerate simplex will be called \textit{geometric} simplex, to stress that only these simplexes really have a geometrical meaning; the degenerate simplexes can be understood as \textit{formal} artifacts introduced for technical (combinatorial) reasons. This becomes clear in the following discussion.

The next essential result, which follows from the commuting properties of degeneracy maps in the definition of simplicial sets provides a way to represent any simplex of a simplicial set in a unique manner.

**Proposition 1.20.** Let \(K\) be a simplicial set. Any \(n\)-simplex \(x \in K^n\) can be expressed in a unique way as a (possibly) iterated degeneracy of a non degenerate simplex \(y\) in the following way:

\[
x = \eta_{j_k} \ldots \eta_{j_1} y
\]

with \(y \in K^r\), \(k = n - r \geq 0\), and \(0 \leq j_1 < \cdots < j_k < n\).

This proposition allows us to encode all the elements (simplexes) of any simplicial set in a generic way, by means of a structure called \textit{abstract simplex}. More concretely, an \textit{abstract simplex} is a pair \((dgop \ gmsm)\) consisting of a sequence of degeneracy maps \(dgop\) (which will be called a \textit{degeneracy operator}) and a geometric simplex \(gmsm\). The indexes in a degeneracy operator \(dgop\) must be in a strictly decreasing order. For instance, if \(\sigma\) is a non degenerate simplex, and \(\sigma'\) is the degenerate simplex \(\eta_1 \eta_2 \sigma\), the corresponding abstract simplexes are respectively \((\emptyset \ \sigma)\) and \((\eta_3 \eta_1 \ \sigma)\), as \(\eta_1 \eta_2 = \eta_3 \eta_1\), due to equality (2) in Definition 1.17.

In a similar way that we defined chain subcomplex we can define the notion of simplicial subcomplex.

**Definition 1.21.** Let \(K = (K^n, \partial, \eta_{n})_{n \in \mathbb{Z}}\) be a simplicial set. \(L = (L^n)_{n \in \mathbb{Z}}\) is a simplicial subcomplex of \(K\) if

- \(L^n \subset K^n\) for all \(n \in \mathbb{Z}\)
- \((L^n, \partial_i|_{L^n}, \eta_i|_{L^n})_{n \in \mathbb{Z}}\) is a simplicial set

Let us show now, other examples of simplicial sets that are (simplicial) models of well-known topological spaces. The following list comes from the simplicial sets that will appear frequently in the rest of the memoir.

**Definition 1.22.** For \(m \geq 1\), the \textit{sphere of dimension} \(m\), \(S^m\), is a simplicial set built as follows. There are just two geometric simplexes: a 0-simplex, let us denote it by \(*\), and an \(m\)-simplex, let us denote it by \(sm\). The faces of \(sm\) are the degeneracies of \(*\); that is to say \(\partial_i(sm) = \eta_{m-1} \eta_{m-2} \ldots \eta_0 \ast\) for all \(0 \leq i \leq m\).
Definition 1.23. Let \( m_1, \ldots, m_n \) natural numbers such that \( m_i \geq 1 \) for all \( 1 \leq i \leq n \), the wedge of spheres of dimensions \( m_1, \ldots, m_n \), \( S^{m_1} \vee \ldots \vee S^{m_n} \), is a simplicial set built as follows. It has the following geometric simplexes: in dimension 0 a 0-simplex, let us denote it by \( \star \), and in dimension \( p \) as many simplexes as the number of \( m_j \), \( 1 \leq i \leq n \), such that \( m_j = p \). The faces of every \( p \)-simplex are the degeneracies of \( \star \); that is to say, let \( x \) a \( p \)-simplex, then \( \partial_i(x) = \eta_{p-1} \eta_{p-2} \ldots \eta_0 \star \) for all \( 0 \leq i \leq p \).

Definition 1.24. For \( n > 0 \), \( p > 1 \) and \( n > 2p - 4 \), the Moore space of dimensions \( p, n \), \( M(\mathbb{Z}/p\mathbb{Z}, n) \), is a simplicial set built as follows. There are just three geometric simplexes: a 0-simplex, let us denote it by \( \star \), an \( n \)-simplex, let us denote it by \( Mn \), and an \( n + 1 \)-simplex, let us denote it by \( Mn' \). The faces of \( Mn \) are the degeneracies of \( \star \); moreover, \( p \) of the faces of \( Mn' \) are identified with \( Mn \) and the rest are the degeneracies of \( \star \).

Definition 1.25. The real projective plane, \( P^\infty \mathbb{R} \), is a simplicial set built as follows. In dimension \( n \), this simplicial set has only one geometric simplex, namely the integer \( n \). The faces of this non degenerate simplex \( n \) are given by the following formulas: \( \partial_0 n = \partial_n n = n - 1 \) and for \( i \neq 0 \) and \( i \neq n \), \( \partial_i n = \eta_{i-1}(n - 2) \).

The real projective plane \( n \)-dimensional, \( P^n \mathbb{R} \), is a simplicial set analogous to the model of \( P^\infty \mathbb{R} \) but without simplexes in dimensions \( m \geq n \).

We can also construct simplicial models of truncated real projective planes. Let \( n > 1 \), \( P^\infty \mathbb{R}/P^n \mathbb{R} \) is a simplicial set analogous to the model of \( P^\infty \mathbb{R} \) but without simplexes in dimensions \( 1 \leq m < n \). The faces of the \( n \)-simplex \( n \) are the degeneracies of the 0-simplex 0.

Let \( n > 1 \) and \( l \geq n \), \( P^l \mathbb{R}/P^{n-1} \mathbb{R} \) is a simplicial set analogous to the model of \( P^l \mathbb{R} \) but without simplexes in dimensions \( 1 \leq m < n \).

The four above definitions are related to simplicial models of initial topological spaces. Moreover, we also have models for topological constructors that are applied to some spaces to obtain new ones.

Definition 1.26. Given two simplicial sets \( K \) and \( L \), the Cartesian product \( K \times L \) is a simplicial set with \( n \)-simplexes:

\[
(K \times L)^n = K^n \times L^n
\]

and for all \( (x, y) \in K^n \times L^n \) the face and degeneracy operators are defined as follows:

\[
\partial_i(x, y) = (\partial_i x, \partial_i y) \quad \text{for} \; 0 \leq i \leq n
\]
\[
\eta_i(x, y) = (\eta_i x, \eta_i y) \quad \text{for} \; 0 \leq i \leq n
\]

Let \( K \) be a simplicial set and \( \star \in K_0 \) a chosen 0-simplex (called the base point). We will also denote by \( \star \) the degenerate simplexes \( \eta_{n-1} \ldots \eta_0 \star \in K_n \) for every \( n \).
1.1 Mathematical preliminaries

Definition 1.27. A simplicial set $K$ is said to be reduced (or 0-reduced) if $K$ has only one 0-simplex. Given $m \geq 1$, $K$ is $m$-reduced if it is reduced and there are not any degenerate simplexes $\forall n, 0 < n \leq m$.

Definition 1.28. Given a reduced simplicial set $X$ with 0-simplex $k_0$, the suspension $\Sigma(X)$ is a simplicial defined as follows. $\Sigma(X)^0 = b_0$; $\Sigma(X)^n$ consists of all symbols $(i, x), i \geq 1, x \in X^{n-i}$, modulo the identification $(i, k_n) = \eta_{n+1} \eta_{n+2} \ldots \eta_0 b_0 = b_{n+i}$, where $k_n = \eta_n \eta_{n-1} \ldots \eta_0 k_0$. The face and degeneracy operators on $\Sigma(X)$ are defined by:

$$
\eta_0 \ldots \eta_0 (j, x) = (i + j, x)
$$
$$
\eta_{i+1} (1, x) = (1, \eta_0 x)
$$
$$
\partial_0 (1, x) = b_n, x \in X^n
$$
$$
\partial_1 (1, x) = b_0, x \in X^0
$$
$$
\partial_{i+1} (1, x) = (1, \partial_0 x), x \in X^n, n > 0
$$

We define inductively $\Sigma^n(X) = \Sigma(\Sigma^{n-1}(X))$ for all $n \geq 1$, $\Sigma^0(X) = X$.

If we impose some condition on the sets of simplicial sets we will obtain new kinds of simplicial objects.

Definition 1.29. An (abelian) simplicial group $G$ is a simplicial set where each $G^n$ is an (abelian, respectively) group and the face and degeneracy operators are group morphisms.

Examples of simplicial groups are provided as follows.

Definition 1.30. Given a reduced simplicial set $X$, the loop space simplicial version of $X$, $G(X)$, is a simplicial set defined as follows: $G^n(X)$ is the free group generated by $X^{n+1}$ with the relations $\{ \eta_0 x; x \in X^n \}$. If $e_n$ is the identity element of $G^n(X)$, we have the canonical application $\tau : X^{n+1} \rightarrow G^n(X)$ defined by:

$$
\tau(y) = \begin{cases} 
e_n & \text{if exists } x \in X^n \text{ such that } y = \eta_0 x, \\ y & \text{otherwise.} \end{cases}
$$

If $\partial, \eta$ are the face and degeneracy operators of $X$, we define the face and degeneracy operators, denoted by $\overline{\partial}, \overline{\eta}$, over the generators of $G^n(X)$ as follows:

$$
\overline{\partial} \tau(x) = [\tau(\partial_0 x)]^{-1} \tau(\partial_1 x)
$$
$$
\overline{\partial}_i \tau(x) = \tau(\partial_{i+1} x) \text{ if } i > 0
$$
$$
\overline{\eta}_i \tau(x) = \tau(\eta_{i+1} x) \text{ if } i \geq 0
$$

In the sequel, we will denote $G(X)$ as $\Omega(X)$. We define inductively $\Omega^n(X) = \Omega(\Omega^{n-1}(X))$ for all $n \geq 1$, $\Omega^0(X) = X$. 
An important example of abelian simplicial groups is the model for an Eilenberg MacLane space.

**Definition 1.31.** An Eilenberg MacLane space of type \((π,n)\) is a simplicial group \(K\) with base point \(e_0 \in K^0\) such that \(π_i(K) = π\) and \(π_i(K) = 0\) if \(i \neq n\) (where \(π_n(K)\) is the \(n\)-th homotopy group), that is, the set of homotopy classes of continuous maps \(f : S^n \to K\) that map a chosen base point \(a \in S^n\) to the base point \(e_0\), a more detailed description and results about homotopy groups can be found in [Hat02]). The simplicial group \(K\) is called a \(K(π,n)\) if it is an Eilenberg MacLane space of type \((π,n)\) and in addition it is minimal.

In order to construct the spaces \(K(π,n)\)'s several methods can be used, although the results are necessarily isomorphic [May67]. Let us consider the following one.

Let \(π\) be an abelian group. First of all, we construct an abelian simplicial group \(K = K(π,0)\) given by \(K^n = π\) for all \(n \geq 0\), and with face and degeneracy operators \(∂_i : K^n = π \to K^{n-1} = π\) and \(η_i : K^n = π \to K^{n+1} = π\), \(0 ≤ i ≤ n\), equal to the identity map of the group \(π\). This abelian simplicial group is the Eilenberg MacLane space of type \((π,0)\). For \(n ≥ 0\), we build recursively \(K(π,n)\) by means of the classifying space constructor.

**Definition 1.32.** Let \(G\) be an abelian simplicial group. The *classifying space* of \(G\), written \(B(G)\), is the abelian simplicial group built as follows. The \(n\)-simplexes of \(B(G)\) are the elements of the Cartesian product:

\[
B(G)^n = G^{n-1} \times G^{n-2} \times \cdots \times G^0.
\]

In this way \(B(G)^0\) is the group which has only one element that we denote by \([\;]\). For \(n ≥ 1\), an element of \(B(G)^n\) has the form \([g_{n-1}, \ldots, g_0]\) with \(g_i \in G^i\). The face and degeneracy operators are given by

\[
η_0[\;] = [e_0], \\
∂_i[\;] = [\;], \quad i = 0, 1, \\
∂_0[g_{n-1}, \ldots, g_0] = [g_{n-2}, \ldots, g_0], \\
∂_i[g_{n-1}, \ldots, g_0] = [g_{i-1}g_{n-1}, \ldots, g_{i}g_{n-i+1}, g_0g_{n-i} + g_{n-i-1}, g_{n-i-2}, \ldots, g_0], \quad 0 < i ≤ n, \\
η_0[g_{n-1}, \ldots, g_0] = [e_n, g_{n-1}, \ldots, g_0], \\
η_i[g_{n-1}, \ldots, g_0] = [η_{i-1}g_{n-1}, \ldots, η_0g_{n-i}, e_{n-i}, g_{n-i-1}, \ldots, o], \quad 0 < i ≤ n
\]

where \(e_n\) denotes the null element of the abelian group \(G^n\).

We define inductively \(B^n(G) = B(B^{n-1}(G))\) for all \(n ≥ 1\), \(B^0(G) = G\).

**Theorem 1.33.** [May67] Let \(π\) be an abelian group and \(K(π,0)\) as explained before. Then \(B^n(K)\) is a \(K(π,n)\).
1.1 Mathematical preliminaries

1.1.2.2 Link between Homological Algebra and Simplicial Topology

Up to now, we have given a brief introduction to Simplicial Topology and Homological Algebra; let us present now the link between these two subjects that will allow us to compute the homology groups of simplicial sets.

**Definition 1.34.** Let $K$ be a simplicial set, we define the *chain complex associated with* $K$, $C_\ast(K) = (C_n(K), d_n)_{n \in \mathbb{N}}$, in the following way:

- $C_n(K) = \mathbb{Z}[K^n]$ is the free $\mathbb{Z}$-module generated by $K^n$. Therefore an $n$-chain $c \in C_n(K)$ is a combination $c = \sum_{i=1}^{m} \lambda_i x_i$ with $\lambda_i \in \mathbb{Z}$ and $x_i \in K^n$ for $1 \leq i \leq m$;
- the differential map $d_n : C_n(K) \to C_{n-1}(K)$ is given by

$$d_n(x) = \sum_{i=0}^{n} (-1)^i \partial_i(x) \text{ for } x \in K^n$$

and it is extended by linearity to the combinations $c = \sum_{i=1}^{m} \lambda_i x_i \in C_n(K)$.

The following statement is an immediate consequence of the previous definition.

**Proposition 1.35.** Let $X, Y$ simplicial sets, where $X$ is a simplicial subcomplex of $Y$, then $C_\ast(X)$ is a chain subcomplex of $C_\ast(Y)$.

From the link between simplicial sets and chain complexes we can define the homology groups of a simplicial set as follows.

**Definition 1.36.** Given a simplicial set $K$, the *$n$-homology group* of $K$, $H_n(K)$, is the $n$-homology group of the chain complex $C_\ast(K)$:

$$H_n(K) = H_n(C_\ast(K))$$

To sum up, when we want to study properties of a compact topological space which admits a triangulation, we can proceed as follows. We can associate a simplicial model $K$ with a topological space $X$ which admits a triangulation. Subsequently, the chain complex $C_\ast(K)$ associated with $K$ can be constructed. Afterwards, properties of this chain complex are computed; for instance, homology groups. Eventually, we can interpret the properties of the chain complex as properties of the topological space $X$.

1.1.3 Effective Homology

As we have seen, a central problem in our context consists in computing *homology groups* of topological spaces. By definition, the homology group of a space $X$ is the one of its associated chain complex $C_\ast(X)$. If $C_\ast(X)$ can be described as a graded
free abelian group with finitely many generators at each degree; then, computing each homology group can be translated to a problem of diagonalizing certain integer matrices (see [Veb31]). So, we can assert that homology groups are computable in this finite type case.

However, things are more interesting when a space $X$ is not of finite type (in the previously invoked sense) but it is known that its homology groups are of finite type. Then, it is natural to study if these homology groups are computable. The effective homology method, introduced in [Ser87] and [Ser94], provides a framework where this computability question can be handled.

In this subsection, we present some definitions and fundamental results about the effective homology method. More details can be found in [RS02] and [RS06].

In the context of effective homology, we can distinguish two different kinds of objects: effective and locally effective chain complexes.

**Definition 1.37.** An effective chain complex is a free chain complex of $\mathbb{Z}$-modules $C_* = (C_n, d_n)_{n \in \mathbb{N}}$ where each group $C_n$ is finitely generated and:

- an algorithm returns a (distinguished) $\mathbb{Z}$-basis in each degree $n$, and
- an algorithm provides the differential maps $d_n$.

If a chain complex $C_* = (C_n, d_n)_{n \in \mathbb{N}}$ is effective, the differential maps $d_n : C_n \to C_{n-1}$ can be expressed as finite integer matrices, and then it is possible to know everything about $C_*$: we can compute the subgroups $\text{Ker} d_n$ and $\text{Im} d_{n+1}$, we can determine whether an $n$-chain $c \in C_n$ is a cycle or a boundary, and in the last case, we can obtain $z \in C_{n+1}$ such that $c = d_{n+1}(z)$. In particular an elementary algorithm computes its homology groups using, for example, the Smith Normal Form technique (for details, see [Veb31]).

**Definition 1.38.** A locally effective chain complex is a free chain complex of $\mathbb{Z}$-modules $C_* = (C_n, d_n)_{n \in \mathbb{N}}$ where each group $C_n$ can have infinite nature, but there exists algorithms such that $\forall x \in C_n$, we can compute $d_n(x)$.

In this case, no global information is available. For example, it is not possible in general to compute the subgroups $\text{Ker} d_n$ and $\text{Im} d_{n+1}$, which can have infinite nature.

In general, we can talk of locally effective objects when only “local” computations are possible. For instance, we can consider a locally effective simplicial set; the set of $n$-simplexes can be infinite, but we can compute the faces of any specific $n$-simplex.

The effective homology technique consists in combining locally effective objects with effective chain complexes. In this way, we will be able to compute homology groups of locally effective objects.

The following notion is one of the fundamental notions in the effective homology method, since it will allow us to obtain homology groups of locally effective chain complexes in some situations.
Definition 1.39. A reduction \( \rho \) (also called contraction) between two chain complexes \( C_\ast \) and \( D_\ast \), denoted in this memoir by \( \rho : C_\ast \Rightarrow D_\ast \), is a triple \( \rho = (f, g, h) \)

\[
\begin{array}{c}
\text{\( h \)} \\
\text{\( C_\ast \)} \quad \text{\( f \)} \\
\text{\( g \)} \\
\text{\( D_\ast \)}
\end{array}
\]

where \( f \) and \( g \) are chain complex morphisms, \( h \) is a graded group morphism of degree +1, and the following relations are satisfied:

1) \( f \circ g = \text{Id}_{D_\ast} \);
2) \( d_C \circ h + h \circ d_C = -g \circ f \);
3) \( f \circ h = 0; \quad h \circ g = 0; \quad h \circ h = 0 \).

The importance of reductions lies in the following fact. Let \( C_\ast \Rightarrow D_\ast \) be a reduction, then \( C_\ast \) is the direct sum of \( D_\ast \) and an acyclic chain complex; therefore the graded homology groups \( H_\ast(C_\ast) \) and \( H_\ast(D_\ast) \) are canonically isomorphic.

Very frequently, the small chain complex \( D_\ast \) is effective; so, we can compute its homology groups by means of elementary operations with integer matrices. On the other hand, in many situations the big chain complex \( C_\ast \) is locally effective and therefore its homology groups cannot directly be determined. However, if we know a reduction from \( C_\ast \) over \( D_\ast \) and \( D_\ast \) is effective, then we are able to compute the homology groups of \( C_\ast \) by means of those of \( D_\ast \).

Given a chain complex \( C_\ast \), a trivial reduction \( \rho = (f, g, h) : C_\ast \Rightarrow C_\ast \) can be constructed, where \( f \) and \( g \) are identity maps and \( h = 0 \).

As we see in the next proposition, the composition of two reductions can be easily constructed.

Proposition 1.40. Let \( \rho = (f, g, h) : C_\ast \Rightarrow D_\ast \) and \( \rho' = (f', g', h') : D_\ast \Rightarrow E_\ast \) be two reductions. Another reduction \( \rho'' = (f'', g'', h'') : C_\ast \Rightarrow E_\ast \) is defined by:

\[
\begin{align*}
    f'' &= f' \circ f \\
    g'' &= g \circ g' \\
    h'' &= h + g \circ h' \circ f
\end{align*}
\]

Another important notion that provides a connection between locally effective homology chain complexes and effective chain complexes is the notion of equivalence.

Definition 1.41. A strong chain equivalence (from now on, equivalence) \( \varepsilon \) between two chain complexes \( C_\ast \) and \( D_\ast \), denoted by \( \varepsilon : C_\ast \Leftrightarrow D_\ast \), is a triple \( (B_\ast, \rho_1, \rho_2) \) where \( B_\ast \) is a chain complex, and \( \rho_1 \) and \( \rho_2 \) are reductions from \( B_\ast \) over \( C_\ast \) and \( D_\ast \) respectively:

\[
\begin{array}{c}
\text{\( C_\ast \)} \quad \text{\( \rho_1 \)} \\
\text{\( \rho_2 \)} \\
\text{\( B_\ast \)} \quad \text{\( \rho_2 \)} \\
\text{\( \rho_1 \)} \\
\text{\( D_\ast \)}
\end{array}
\]
Very frequently, $D_*$ is effective; so, we can compute its homology groups by means of elementary operations with integer matrices. On the other hand, in many situations both $C_*$ and $B_*$ are locally effective and therefore their homology groups cannot directly be determined. However, if we know a reduction from $B_*$ over $C_*$, a reduction from $B_*$ over $D_*$, and $D_*$ is effective, then we are able to compute the homology groups of $C_*$ by means of those of $D_*$.  

Once we have introduced the notion of equivalence, it is possible to give the definition of object with effective homology, which is the fundamental idea of the effective homology technique. These objects will allow us to compute homology groups of locally effective objects by means of effective chain complexes.

**Definition 1.42.** An object with effective homology $X$ is a quadruple $(X, C_*(X), H C_*, \varepsilon)$ where:

- $X$ is a locally effective object;
- $C_*(X)$ is a (locally effective) chain complex associated with $X$, that allows us to study the homological nature of $X$;
- $H C_*$ is an effective chain complex;
- $\varepsilon$ is an equivalence $\varepsilon : C_*(X) \Leftrightarrow H C_*$.  

Then, the graded homology groups $H_*(X)$ and $H_*(H C_*)$ are canonically isomorphic, then we are able to compute the homology groups of $X$ by means of those of $H C_*$.  

The main problem now is the following one: given a chain complex $C_* = (C_n, d_n)_{n \in \mathbb{N}}$, is it possible to determine its effective homology? We must distinguish three cases:

- First of all, if a chain complex $C_*$ is by chance effective, then we can choose the trivial effective homology: $\varepsilon$ is the equivalence $C_* \Leftrightarrow C_* \Rightarrow C_*$, where the two components $\rho_1$ and $\rho_2$ are both the trivial reduction on $C_*$.  

- In some cases, some theoretical results are available providing an equivalence between some chain complex $C_*$ and an effective chain complex. Typically, the Eilenberg MacLane space $K(\mathbb{Z}, 1)$ has the homotopy type of the circle $S^1$ and a reduction $C_*(K(\mathbb{Z}, 1)) \Rightarrow C_*(S^1)$ can be built.  

- The most important case: let $X_1, \ldots, X_n$ be objects with effective homology and $\Phi$ a constructor that produces a new space $X = \Phi(X_1, \ldots, X_n)$ (for example, the Cartesian product of two simplicial sets, the classifying space of a simplicial group, etc). In natural “reasonable” situations, there exists an effective homology version of $\Phi$ that allows us to deduce a version with effective homology of $X$, the result of the construction, from versions with effective homology of the arguments $X_1, \ldots, X_n$. For instance, given two simplicial sets $K$ and $L$ with effective homology, then the Cartesian product $K \times L$ is an object with effective homology too; this is obtained by means of the Eilenberg-Zilber Theorem, see [RS06].
Two of the most basic (in the sense of fundamental) results in the effective homology method are the two perturbation lemmas. The main idea of both lemmas is that given a reduction, if we perturb one of the chain complexes then it is possible to perturb the other one so that we obtain a new reduction between the perturbed chain complexes. The first theorem (the Easy Perturbation Lemma) is very easy, but it can be useful. The Basic Perturbation Lemma is not trivial at all. It was discovered by Shih Weishu [Shi62], although the abstract modern form was given by Ronnie Brown [Bro67].

**Definition 1.43.** Let $C_n = (C_n, d_n)_{n \in \mathbb{N}}$ be a chain complex. A perturbation $\delta$ of the differential $d$ is a collection of group morphisms $\delta = \{\delta_n : C_n \to C_{n-1}\}_{n \in \mathbb{N}}$ such that the sum $d + \delta$ is also a differential, that is to say, $(d + \delta) \circ (d + \delta) = 0$.

The perturbation $\delta$ produces a new chain complex $C'_n = (C_n, d_n + \delta_n)_{n \in \mathbb{N}}$; it is the perturbed chain complex.

**Theorem 1.44 (Easy Perturbation Lemma, EPL).** Let $C_n = (C_n, d_C)_{n \in \mathbb{N}}$ and $D_n = (D_n, d_D)_{n \in \mathbb{N}}$ be two chain complexes, $\rho = (f, g, h) : C_n \Rightarrow D_n$ a reduction, and $\delta_D$ a perturbation of $d_D$. Then a new reduction $\rho' = (f', g', h') : C_n \Rightarrow D_n$ can be constructed where:

1) $C'_n$ is the chain complex obtained from $C_n$ by replacing the old differential $d_C$ by a perturbed differential $d_C + g \circ \delta_D \circ f$;

2) the new chain complex $D'_n$ is obtained from the chain complex $D_n$ only by replacing the old differential $d_D$ by $d_D + \delta_D$;

3) $f' = f$;

4) $g' = g$;

5) $h' = h$.

The perturbation $\delta_D$ of the small chain complex $D_n$ is naturally transferred (using the reduction $\rho$) to the big chain complex $C_n$, obtaining in this way a new reduction $\rho'$ between the perturbed chain complexes. On the other hand, if we consider a perturbation $d_C$ of the top chain complex $C_n$, in general it is not possible to perturb the small chain complex $D_n$ so that there exists a reduction between the perturbed chain complexes. As we will see, we need an additional hypothesis.

**Theorem 1.45 (Basic Perturbation Lemma, BPL).** [Bro67] Let us consider a reduction $\rho = (f, g, h) : C_n \Rightarrow D_n$ between two chain complexes $C_n = (C_n, d_C)_{n \in \mathbb{N}}$ and $D_n = (D_n, d_D)_{n \in \mathbb{N}}$, and $\delta_C$ a perturbation of $d_C$. Furthermore, the composite function $h \circ \delta_C$ is assumed locally nilpotent, in other words, given $x \in C_n$ there exists $m \in \mathbb{N}$ such that $(h \circ \delta_C)^m(x) = 0$. Then a new reduction $\rho' = (f', g', h') : C_n \Rightarrow D_n$ can be constructed where:
1) $C'_\ast$ is the chain complex obtained from the chain complex $C_\ast$ by replacing the old differential $d_C$ by $d_C + \delta_C$;

2) the new chain complex $D'_\ast$ is obtained from $D_\ast$ by replacing the old differential $d_D$ by $d_D + \delta_D$, with $\delta_D = f \circ \delta_C \circ \phi \circ g = f \circ \psi \circ \delta_C \circ g$;

3) $f' = f \circ \psi = f \circ (\text{Id}_{C_\ast} - \delta_C \circ \phi \circ h)$;

4) $g' = \phi \circ g$;

5) $h' = \phi \circ h = h \circ \psi$;

with the operators $\phi$ and $\psi$ defined by

$$\phi = \sum_{i=0}^{\infty} (-1)^i (h \circ \delta_C)^i$$
$$\psi = \sum_{i=0}^{\infty} (-1)^i (\delta_C \circ h)^i = \text{Id}_{C_\ast} - \delta_C \circ \phi \circ h,$$

the convergence of these series being ensured by the local nilpotency of the compositions $h \circ \delta_C$ and $\delta_C \circ h$.

It is worth noting that the effective homology method is not only a theoretical method but it has also been implemented in a software system called Kenzo [DRSS98]. In this system, the BPL is a central result and has been intensively used.

### 1.2 The Kenzo system

Kenzo is a 16000 lines program written in Common Lisp [Gra96], devoted to Symbolic Computation in Algebraic Topology. It was developed by Francis Sergeraert and some co-workers, and is www-available (see [DRSS98] for documentation and details). It works with the main mathematical structures used in Simplicial Algebraic Topology, [HW67], (chain complexes, differential graded algebras, simplicial sets, morphisms between these objects, reductions and so on) and has obtained some results (for example, homology groups of iterated loop spaces of a loop space modified by a cell attachment, see [Ser92]) which have not been confirmed nor refuted by any other means.

The fundamental idea of the Kenzo system is the notion of object with effective homology combined with functional programming. By using functional programming, some Algebraic Topology concepts are encoded in the form of algorithms. In addition, not only known algorithms were implemented but also new methods were developed to transform the main “tools” of Algebraic Topology, mainly the spectral sequences (which are not algorithmic in the traditional organization), into actual computing methods.
The computation process in the Kenzo system requires a great amount of algebraic structures and equivalences to be built, and thus a lot of resources.

This section is devoted to present some important features of Kenzo that will be relevant onward.

### 1.2.1 Kenzo mathematical structures

The data structures of the Kenzo system are organized in two layers. As algebraically modeled in [LPR03, DLR07], the first layer is composed of algebraic data structures (chain complexes, simplicial sets, and so on) and the second one of standard data structures (lists, trees, and so on) which are representing elements of data from the first layer.

Here we give an overview of how mathematical structures (the first layer) are represented in the Kenzo system, using object oriented (CLOS) and functional programming features simultaneously.

Most often, an object of some type in mathematics is a structure with several components, frequently of functional nature. Let us consider the case of a user who wants to handle groups; this simple particular case is sufficient to understand how CLOS gives the right tools to process mathematical structures.

We can define a GRP class whose instances correspond to concrete groups as follows.

```lisp
> (DEFCLASS GRP ()
  ((elements :type list :initarg :elements :reader elements)
   (mult :type function :initarg :mult :reader multi)
   (inv :type function :initarg :inv :reader inv)
   (nullel :type function :initarg :nullel :reader nullel)
   (cmp :type function :initarg :cmp :reader cmp)))

#<STANDARD-CLASS GRP>
```

In this organization, a group is made of five slots which represent a list of the elements of the group (elements), the product of two elements of the group (mult), the inverse of an element of the group (inv), the identity element of the group (nullel) and a comparison test (cmp) between the elements of the group. It is worth noting that mult, inv, nullel and cmp slots are functional slots.

The most important difference between the mathematical definition and its implementation is that no axiomatic information appears in CLOS. Kenzo, being a system for computing, does not need any information about the properties of the operations, only their behavior is relevant. As a consequence of this, abelian groups can be implemented in Common Lisp similarly to groups, even when their mathematical definitions differ.

Let us construct now the group \( \mathbb{Z}/2\mathbb{Z} \); that is to say, the cyclic group of dimension 2. In our context we define the comparison, the product and the inverse functions with
the help of the mod function, a predefined Lisp function.

In this way, we can define mathematical structures in Common Lisp using object oriented (CLOS) and functional programming features. The interested reader can consult [Ser01] where a detailed explanation about how defining mathematical structures in Common Lisp is presented.

The previously explained organization is moved to the Kenzo context in order to define the main mathematical structures used in Simplicial Algebraic Topology, [HW67]. Figure 1.2 shows the Kenzo class diagram where each class corresponds to the respective mathematical structure.

The lefthand part of the class diagram is made of the main mathematical categories that are used in combinatorial Algebraic Topology. As we said previously, a chain complex is a graded differential module; an algebra is a chain complex with a compatible multiplicative structure, the same for a coalgebra but with comultiplicative structure. If

![Figure 1.2: Kenzo class diagram](image-url)
a multiplicative and a comultiplicative structures are added and if they are compatible with each other in a natural sense, then it is a Hopf algebra, and so on. The righthand part of the class diagram is made of the operations over the mathematical structures of the lefthand. It is worth noting that all the mathematical structures in the Kenzo system are graded structures.

The following class definition corresponds to the simplest algebraic structure implemented in Kenzo, free chain complexes:

```
(DEFCLASS CHAIN-COMPLEX ()
  ((cmpr :type cmprf :initarg :cmpr :reader cmpr1)
   (basis :type basis :initarg :basis :reader basis1)
   ;; Base Generator
   (bsgn :type gnrt :initarg :bsgn :reader bsgn)
   ;; Differential
   (dffr :type morphism :initarg :dffr :reader dffr1)
   ;; Ground Module
   (grmd :type chain-complex :initarg :grmd :reader grmd)
   ;; Effective Homology
   (efhm :type homotopy-equivalence :initarg :efhm :reader efhm)
   ;; Identification Number
   (idnm :type fixnum :initform (incf *idnm-counter*) :reader idnm)
   ;; Origin
   (orgn :type list :initarg :orgn :reader orgn)))
```

The relevant slots are cmpr, a function coding the equality between the generators of the chain complex; basis, the function defining the basis of each group of $n$-chains, or the keyword :locally-effective if the chain complex is not effective; dffr, the differential morphism, which is an instance of the class MORPHISM; efhm, which stores information about the effective homology of the chain complex; and orgn, is used to keep record of information about the object.

The class CHAIN-COMPLEX is extended by inheritance with new slots, obtaining more elaborate structures. For instance, extending it with an aprd (algebra product) slot, we obtain the ALGEBRA class. Multiple inheritance is also available; for example, the class SIMPLICIAL-GROUP is obtained by inheritance from the classes KAN and HOPF-ALGEBRA.

It is worth emphasizing here that simplicial sets have also been implemented as a subclass of CHAIN-COMPLEX. To be precise, the class SIMPLICIAL-SET inherits from the class COALGEBRA, which is a direct subclass of CHAIN-COMPLEX, with a slot cprd (the coproduct). The class SIMPLICIAL-SET has then one slot of its own: face, a Lisp function computing any face of a simplex of the simplicial set. The basis is in this case (when working with effective objects) a function associating to each dimension $n$ the list of non degenerate $n$-simplexes, and the differential map of the associated chain complex is given by the alternate sum of the faces, where the degenerate simplexes are cancelled.
1.2.2 Kenzo way of working

In Kenzo there is a one higher-level objective: compute groups associated with topological spaces. This main objective can be broken in two actions: (1) computing groups, and (2) constructing spaces. Note that the second task is necessary to carry out the first one.

When a user has decided to construct a space in Kenzo, he should decide which type he wants to build: a simplicial set, a simplicial group and so on; namely, an object of one of the types of the lefthand part of the class diagram of Figure 1.2. Therefore, the user has to construct an instance of one of those classes.

As this task can be quite difficult for a non expert user, Kenzo provides useful functions to create interesting objects of regular usage, which belong to four types: chain complexes, simplicial sets, simplicial groups and abelian simplicial groups.

These functions can be split in two different kinds: (1) functions to construct initial spaces and, (2) functions to construct spaces from other ones applying topological constructors.

The following elements, gathered by the types of the constructed object, represent the main spaces that can be constructed in Kenzo from scratch (that is to say, which belong to the first kind):

- **Chain Complexes:**
  - *Unit chain complex:* the \texttt{zcc} function, with no arguments, constructs the unit chain complex, see Example 1.7.
  - *Circle:* the \texttt{circle} function, with no arguments, constructs the circle chain complex, see Example 1.7.

- **Simplicial Sets:**
  - *Standard simplicial set:* the \texttt{delta} function, with a natural number $n$ as argument, constructs $\Delta[n]$, see Definition 1.18.
  - *Sphere:* the \texttt{sphere} function, with a natural number $n$ as argument, constructs $S^n$, see Definition 1.22.
  - *Sphere Wedge:* the \texttt{sphere-wedge} function, with a sequence of natural numbers $n_1, \ldots, nk$ as arguments, constructs $S^{n_1} \cup \ldots \cup S^{nk}$, see Definition 1.23.
  - *Moore space:* the \texttt{moore} function, with two natural number $n, p$ as arguments, constructs $M(\mathbb{Z}/n\mathbb{Z}, p)$, see Definition 1.24.
  - *Projective space:* the \texttt{r-proj-space} function has two optional arguments $k$ and $l$. If neither $k$ and $l$ are provided, then the function constructs $\mathbb{R}^\infty$. If $k$ is provided but $l$ is not, then the function constructs $\mathbb{R}^\infty/\mathbb{R}^{k-1}$. If both $k$ and $l$ are provided, then the function constructs $\mathbb{R}^l/\mathbb{R}^{k-1}$. In the latter case, if $k = 1$, then the function constructs $\mathbb{R}^l$; see Definition 1.25.
Finite simplicial set: the `build-finite-ss` function, with a list of lists as argument, constructs a simplicial set, see [DRSS98].

- Abelian Simplicial Group:
  - *Eilenberg MacLane space* type $(\mathbb{Z}, n)$: the `k-z` function, with a natural number $n$ as argument, constructs $K(\mathbb{Z}, n)$, see Definition 1.31.
  - *Eilenberg MacLane space* type $(\mathbb{Z}/2\mathbb{Z}, n)$: the `k-z2` function, with a natural number $n$ as argument, constructs $K(\mathbb{Z}/2\mathbb{Z}, n)$, see Definition 1.31.

On the contrary, the following elements, gathered by types, represent the main spaces that can be constructed in Kenzo from other spaces applying topological constructors (that is to say, which belong to the second kind):

- Chain Complexes:
  - *Tensor Product*: the `tnsr-prdc` function, with two chain complexes $C_\ast, D_\ast$ as arguments, constructs the tensor product $C_\ast \otimes D_\ast$, see Definition 1.10.

- Simplicial Sets:
  - *Cartesian product*: the `crts-prdc` function, with two simplicial sets $K, L$ as arguments, constructs the Cartesian product $K \times L$, see Definition 1.26.
  - *Suspension*: the `suspension` function, with a simplicial set $X$ and a natural number $n$ as arguments, constructs the suspension $\Sigma^n(X)$, see Definition 1.28.

- Simplicial Group:
  - *Loop space*: the `loop-space` function, with a simplicial set $X$ and a natural number $n$ as arguments, constructs the loop space $\Omega^n(X)$, see Definition 1.30.
  - *Classifying space*: the `classifying` function, with a simplicial group $X$, constructs the classifying space $B(X)$, see Definition 1.32.

Eventually, once we have constructed some spaces in our Kenzo session, the Kenzo user can perform computations. Namely, the `homology` function with a chain complex $X$ (or an instance of one of its subclasses: simplicial set, simplicial group and so on) and a natural number $n$ as arguments computes $H_n(X)$.

To sum up, a simplification of the way of working with Kenzo is as follows. As a first step, the user constructs some initial spaces by means of some built-in Kenzo functions (as spheres, Moore spaces, Eilenberg MacLane spaces and so on); then, in a second step, he constructs new spaces by applying topological constructions (as Cartesian products, loop spaces, and so on); as a third, and final, step, the user asks Kenzo for computing the homology groups of the spaces. Let us remark that this kind of interaction does not fully cover all the Kenzo capabilities, but it is just a simplification.
1.2.3 Kenzo in action

Let us show a didactic example to illustrate the interaction with the Kenzo program. The homology group $H_5(\Omega^3(M(\mathbb{Z}/2\mathbb{Z}, 4)))$ is “in principle” reachable thanks to old methods, see [CM95], but experience shows even the most skilful topologists meet some difficulties to determine it, see [RS02]. With the Kenzo program, you construct the Moore space $M(\mathbb{Z}/2\mathbb{Z}, 4)$ in the following way:

```lisp
> (setf m4 (moore 2 4)) ✠
[K1 Simplicial-Set]
```

A Kenzo display must be read as follows. The initial `>` is the Lisp prompt of this Common Lisp implementation. The user types out a Lisp statement, here `(setf m4 (moore 2 4))` and the maltese cross ✠ (in fact not visible on the user screen) marks in this text the end of the Lisp statement, just to help the reader: the right number of closing parentheses is reached. The Return key then asks Lisp to evaluate the Lisp statement. Here the Moore space $M(\mathbb{Z}/2\mathbb{Z}, 4)$ is constructed by the Kenzo function `moore`, taking into account of the arguments 2 and 4, and this Moore space is assigned to the Lisp symbol `m4` for later use. Also evaluating a Lisp statement returns an object, the result of the evaluation, in this case the Lisp object implementing the Moore space, displayed as `[K1 Simplicial-Set]`, that is, the Kenzo object #1, a `Simplicial-Set`. The internal structure of this object, made of a rich set of data, in particular many functional components, is not displayed. The identification number printed by Kenzo allows the user to recover the whole object by means of a function called simply `k` (for instance, the evaluation of `(k 1)` returns the Moore space $M(\mathbb{Z}/2\mathbb{Z}, 4)$, in our running example). In addition, another function (called `orgn` and which is one of the slots of the `Chain-Complex` class) allows the user to obtain the origin of the object (i.e. from which function and with which arguments has been produced), and thus the printed information is enough to get a complete control of the different objects built with Kenzo.

It is then possible to construct the third loop space of the Moore space, $\Omega^3(M(\mathbb{Z}/2\mathbb{Z}, 4))$, as a simplicial group.

```lisp
> (setf o3m4 (loop-space m4 3)) ✠
[K30 Simplicial-Group]
```

The combinatorial version of the loop space is highly infinite: it is a combinatorial version of the space of continuous maps $S^3 \to M(\mathbb{Z}/2\mathbb{Z}, 4)$, but functionally encoded as a small set of functions in a `Simplicial-Group` object.

Eventually, the user can compute the fifth homology group of this space.
1.2 The Kenzo system

This result must be interpreted as stating \( H_5(\Omega^3(M(\mathbb{Z}/2\mathbb{Z}, 4))) = \mathbb{Z}_2^5 \). In this way, Kenzo computes the homology groups of complicated spaces. This is due to the Kenzo implementation of the effective homology method which is explained in the following subsection.

1.2.4 Effective Homology in Kenzo

As we have previously said, the central idea of the Kenzo system is the notion of object with effective homology. In this subsection, we are going to show how this notion explained in Subsection 1.1.3 is used in Kenzo.

As we stated in Subsection 1.1.3, the main problem is the following one: given an object, determine its effective homology version. Three cases have been distinguished.

First of all, let an object \( X \) if the chain complex \( C_*(X) \) is by chance effective, then we can choose the trivial effective homology: \( \varepsilon \) is the equivalence \( C_*(X) \iff C_*(X) \Rightarrow C_*(X) \), where both reductions are the trivial reduction on \( C_\ast \). This situation happens, for instance, in the case of the sphere \( S^3 \) (we consider a fresh Kenzo session).

We can ask for the effective homology of \( S^3 \) as follows:

\[
> (\text{efhm } s3) \not\!
\]

An homotopy equivalence is automatically constructed by Kenzo where both reductions are the trivial reduction on \( C_\ast(S^3) \) (let us note that the \( K1 \) object not only represents the simplicial set \( S^3 \) but also the chain complex \( C_\ast(S^3) \) due to the heritage relationship between simplicial sets and chain complexes). In this case, the effective homology technique does not provide any additional tool to the computation of the homology groups of \( S^3 \). On the contrary, we will see the power of this technique when the
initial space $X$ is locally effective.

The second feasible situation happened when given a locally effective object $X$ some theoretical result was available providing an equivalence between the chain complex $C_*(X)$ and an effective chain complex.

According to Subsubsection 1.1.2.1, a simplicial model of the Eilenberg MacLane space $K(Z, 1)$ is defined by $K(Z, 1)_n = Z^n$; an infinite number of simplexes is required in every dimension $n \geq 1$; that is, we have a locally effective object. This does not prevent such an object from being installed and handled by Kenzo.

> (setf kz1 (k-z 1)) ✗

[K10 Abelian-Simplicial-Group]

The $k$-$z$ Kenzo function construct the standard Eilenberg MacLane space. In ordinary mathematical notation (as seen in Subsubsection 1.1.2.1), a 3-simplex of $kz1$ could be for example $[3, 5, -5]$, denoted by $(3 \ 5 \ -5)$ in Kenzo. The faces of this simplex can be determined as follows.

> (dotimes (i 4) (print (face kz1 i 3 '(3 5 -5)))) ✗
<AbSm - (5 -5)>
<AbSm - (8 -5)>
<AbSm 1 (3)>
<AbSm - (3 5)>
nil

The faces are computed as explained in Subsubsection 1.1.2.1. Then, local computations are possible, so, the object $kz1$ is locally effective. But no global information is available. For example, if we try to obtain the list on non degenerate simplexes in dimension 3, we obtain an error.

> (basis kz1 3) ✗

Error: The object [K10 Abelian-Simplicial-Group] is locally-effective

This basis in fact is $Z^3$, an infinite set whose element list cannot be explicitly stored nor displayed. So, the homology groups of $kz1$ cannot be elementarily computed. However, $K(Z, 1)$ has the homotopy type of the circle $S^1$ and the Kenzo program knows this fact.

> (efhm kz1) ✗

[K31 Homotopy-Equivalence K10 <= K10 => K25]

A reduction $K10 = K(Z, 1) \Rightarrow k25$ is constructed by Kenzo. What is $k25$?
K25 is the expected object, the circle $S^1$ which is an effective chain complex; so we can compute its homology groups by means of tradicional methods. Therefore, we can compute the homology groups of the space $K(\mathbb{Z}, 1)$ by means of the effective homology method.

Eventually, the last situation happened when given $X_1, \ldots, X_n$ objects with effective homology and $\Phi$ a constructor that produced a new space $X = \Phi(X_1, \ldots, X_n)$, we wanted to compute the effective homology version of $X$. Let us present an example.

The Cartesian product of two locally effective simplicial sets produces another locally effective simplicial set.

So, the homology groups of $kz1xkz1$ cannot be elementarily computed. However, Kenzo is able to construct an equivalence between this object and an effective chain complex.

An equivalence $K15 = K(\mathbb{Z}, 1) \times K(\mathbb{Z}, 1) \Leftrightarrow K53 \Rightarrow K43$ is constructed by Kenzo. What is $K43$?

The object $K43$ is the tensor product of two circles $C_*(S^1) \otimes C_*(S^1)$ (the reduction is obtained from the Eilenberg-Zilber Theorem, see [RS06]), an effective chain complex; so we can compute its homology groups by means of tradicional methods. Therefore, we can compute the homology groups of the space $K(\mathbb{Z}, 1) \times K(\mathbb{Z}, 1)$ by means of the effective homology method.

It is worth noting that a Kenzo user does not need to explicitly construct the equivalence to compute the homology groups of a locally effective object. This task is automatically performed by the Kenzo system which constructs the necessary objects without any additional help.
For instance, let us consider a fresh Kenzo session where we have constructed the space $\Omega^3(M(\mathbb{Z}/2\mathbb{Z}, 4))$, the example of the previous subsection:

```lisp
> (setf m4 (moore 2 4)) ✧
[K1 Simplicial-Set]
> (setf o3m4 (loop-space m4 3)) ✧
[K30 Simplicial-Group]
```

At this moment, we can check the number of objects constructed in Kenzo (this information is stored in a global variable called `*idnm-counter*').

```lisp
> *idnm-counter* ✧
41
```

Subsequently, after computing the third homology, we ask again the number of objects constructed in Kenzo and we obtain the following result.

```lisp
> (homology o3m4 3) ✧
Homology in dimension 3 :
Component $\mathbb{Z}/4\mathbb{Z}$
Component $\mathbb{Z}/2\mathbb{Z}$
> *idnm-counter* ✧
404
```

This means that Kenzo has constructed 363 intermediary objects in order to compute the homology groups of $\Omega^3(M(\mathbb{Z}/2\mathbb{Z}, 4))$; namely, in order to construct an equivalence between the locally effective object $\Omega^3(M(\mathbb{Z}/2\mathbb{Z}, 4))$ and an effective chain complex.

### 1.2.5 Memoization in Kenzo

The Kenzo program is certainly a functional system. It is frequent that several thousands of functions are present in memory, each one being dynamically defined from other ones, which in turn are defined from other ones, and so on. In this quite original situation, the same calculations are frequently asked again. To avoid repeating these calculations, it is better to store the results and to systematically examine for each calculation whether the result is already available (memoization strategy).

As a consequence, the state of a space evolves after it has been used in a computation (of a homology group, for instance). Thus, the time needed to compute, let us say, a homology group, depends on the concrete state of the space involved in the calculation (in the more explicit case, to re-calculate the homology group of a space could be negligible in time, even if in the first occasion this was very time consuming).

Let us shown an example, of the Kenzo memoization. We want to compute the fifth
homology group of the space $\Omega^3(S^4 \times S^4)$ in a fresh Kenzo session and see how much time this computation takes (using the time function); therefore, we proceed as usual.

The first time that we compute $H_5(\Omega^3(S^4 \times S^4))$, Kenzo takes almost 20 minutes to obtain the result. However, if we ask again for the same computation:

in this case Kenzo only needs 4 minutes. It is worth noting that Kenzo does not store the final result (that is to say, the group $H_5(\Omega^3(S^4 \times S^4))$) but intermediary computations related to the differential of the generators of the space. Then, when a Kenzo user asks a computation previously computed, Kenzo does not simply look up and returned it, but it uses some previously stored computations to calculate the result faster.

Moreover, it is very important not to have several copies of the same function; otherwise it is impossible for one copy to guess some calculation has already been done by another copy. This is a very important question in Kenzo, so that the following idea has been used. Each Kenzo object has a rigorous definition, stored as a list in the orgn slot of the object (orgn stands for origin of the object). This is the main reason of the top class kenzo-object: making this process easier. The actual definition of the kenzo-object class is:

Then, when any kenzo-object is to be considered, its definition is constructed and the program firstly looks at *k-list* (a list which stores the already constructed kenzo-object instances) whether some object corresponding to this definition already exists; if yes, no kenzo-object is constructed, the already existing one is simply returned. Look at this small example where we construct the second loop space of $S^3$, then the first loop space, and then again the second loop space. In fact the initial construction of the second loop space required the first loop space, and examining the identification...
number $K??$ of these objects shows that when the first loop space is later asked for, Kenzo is able to return the already existing one.

```lisp
> (setf s3 (sphere 3)) ✘
[K372 Simplicial-Set]
> (setf o2s3 (loop-space s3 2)) ✘
[K380 Simplicial-Group]
> (setf os3 (loop-space s3 1)) ✘
[K374 Simplicial-Group]
> (setf o2s3-2 (loop-space s3 2)) ✘
[K380 Simplicial-Group]
> (eq o2s3 o2s3-2) ✘
T
```

The last statement shows the symbols $o2s3$ and $o2s3-2$ points to the same machine address. In this way we are sure any kenzo-object has no duplicate, so that the memory process for the values of numerous functions cannot miss an already computed result.

### 1.2.6 Reduction degree

Working with Kenzo, a constructor for the space $X$ has associated a number $g(X) \in \mathbb{Z}$. When, we apply an operation $O$ over the space $X$, a set of rules are used to compute $g(O(X))$. We will call reduction degree of $X$ to this number, $g(X)$, which is attached to our concrete representation of the spaces. This number is a lower bound of the simply connectedness degree; that is to say, the homotopy groups of the space $X$ are null at least from 1 to $g(X)$. We list as follows the set of rules for the initial spaces and topological operators which are going to be employed in this memoir.

**Tensor product** ($X \otimes Y$): $g(X \otimes Y) = \min\{g(X), g(Y)\}$.

**Suspension** ($\Sigma^n(X)$): $g(\Sigma^n(X)) = \begin{cases} g(X) + n & \text{if } g(X) \geq 0 \\ g(X) & \text{if } g(X) < 0 \end{cases}$

**Sphere** ($S^n$): $g(S^n) = n - 1$.

**Moore space** ($M(\mathbb{Z}/p\mathbb{Z}, n)$): $g(M(\mathbb{Z}/p\mathbb{Z}, n)) = n - 1$.

**Standard simplicial set** ($\Delta^n$): $g(\Delta^n) = 0$.

**Sphere wedge** ($S^{n_1} \vee S^{n_2} \vee \ldots \vee S^{n_k}$):

$$g(S^{n_1} \vee S^{n_2} \vee \ldots \vee S^{n_k}) = \min\{g(S^{n_1}), g(S^{n_2}), \ldots, g(S^{n_k})\}.$$  

**Projective space** ($P^\infty \mathbb{R}$): $g(P^\infty \mathbb{R}) = 0$.

**Projective space** ($P^\infty \mathbb{R}/P^n \mathbb{R}$): $g(P^\infty \mathbb{R}/P^n \mathbb{R}) = n - 1$. 

1.2 The Kenzo system

Projective space \((\mathbb{P}^l \mathbb{R})\): \(g(\mathbb{P}^l \mathbb{R}) = 0\).

Projective space \((\mathbb{P}^l \mathbb{R}/\mathbb{P}^n \mathbb{R})\): \(g(\mathbb{P}^l \mathbb{R}/\mathbb{P}^n \mathbb{R}) = n - 1\).

\(K(\mathbb{Z}, n)\): \(g(K(\mathbb{Z}, n)) = n - 1\).

\(K(\mathbb{Z}/2\mathbb{Z}, n)\): \(g(K(\mathbb{Z}/2\mathbb{Z}, n)) = n - 1\).

Loop space \((\Omega^n(X))\): \(g(\Omega^n(X)) = g(X) - n\).

Cartesian product \((X \times Y)\): \(g(X \times Y) = \min\{g(X), g(Y)\}\).

Classifying space \((B^n(X))\): \(g(B^n(X)) = \begin{cases} g(X) + n & \text{if } g(X) \geq 0 \\ g(X) & \text{if } g(X) < 0 \end{cases}\)

The reduction degree of a space \(X\) provides us information about the capabilities of the Kenzo system to obtain the homology groups of \(X\). When \(g(X) < 0\), a Kenzo attempt to compute the homology groups of \(X\) will raise an error; otherwise the Kenzo system can compute the homology groups of \(X\).

For instance, the reduction degree of \(\Omega^2 S^2\) is \(-1\); then, let us show what happens if we try to compute the third homology group of this space.

```lisp
> (setf s2 (sphere 2)) ✗
[K1 Simplicial-Set]
> (setf o2s2 (loop-space s2 2)) ✗
[K18 Simplicial-Group]
> (homology o2s2 3) ✗
Error: ‘NIL’ is not of the expected type ‘NUMBER’
[condition type: TYPE-ERROR]
```

As we can see an error is produced; then, the Kenzo system cannot compute the homology groups for \(\Omega^2(S^2)\).

1.2.7 Homotopy groups in Kenzo

Up to now, we have been working with one of the most important algebraic invariants in Algebraic Topology: homology groups. We can wonder what happens with the other main algebraic invariant: homotopy groups.

The \(n\)-homotopy group of a topological space \(X\) with a base point \(x_0\) is defined as the set of homotopy classes of continuous maps \(f : S^n \to X\) that map a chosen base point \(a \in S^n\) to the base point \(x_0 \in X\). A more detailed description and results about homotopy groups can be found in [Hat02, May67].

Homotopy groups were defined by Hurewicz in [Hur35] and [Hur36] as a generalization of the fundamental group [Poi95]. It is worth noting that except in special cases,
homotopy groups are hard to be computed. For instance, whereas the homology groups of spheres are easily computed, computing their homotopy groups remains as a difficult subject, see [Tod62, Mah67, Rav86].

For the general case, Edgar Brown published in [Bro57] a theoretical algorithm for the computation of homotopy groups of simply connected spaces such that their homology groups are of finite type. However Brown himself explained that his “algorithm” has not practical use.

An interesting algorithm based on the effective homology theory was developed by P. Real, see [Rea94]. In that paper, an algorithm that computes the homotopy groups of a 1-reduced simplicial set was explained. Here, we just state the algorithm.

**Algorithm 1.46 ([Rea94]).**

*Input*: a 1-reduced simplicial set with effective homology $X$ and a natural number $n$ such that $n \geq 1$.

*Output*: the $n$-th homotopy group of the underlying simplicial set $X$.

This algorithm is based on the Whitehead tower process [Hat02], a method which allows one to reach any homotopy group of a 1-reduced simplicial set.

It is worth noting that Kenzo, in spite of not providing a function called homotopy, like in the case of homology groups, implements all the necessary tools to use the algorithm presented in [Rea94]. A detailed explanation about how to use this method in Kenzo was explained in Chapter 21 of the Kenzo documentation [DRSS98]. However, it is worth noting that in the current Kenzo version, homotopy groups of a 1-reduced simplicial set $X$ can only be computed if the first non null homology group of $X$ is $\mathbb{Z}$ or $\mathbb{Z}/2\mathbb{Z}$; this is due to the fact that the algorithm presented in [Rea94] needs the calculation of homology groups of Eilenberg-MacLane spaces $K(G, n)$, and in the original Kenzo distribution only the homology of spaces $K(\mathbb{Z}, n)$ and $K(\mathbb{Z}/2\mathbb{Z}, n)$ are built-in.

### 1.3 ACL2

The ACL2 Theorem Prover has been used to verify the correctness of some Kenzo programs, which will be presented in chapters 5 and 6. This section is devoted to provide a brief description of ACL2. In spite of being a glimpse introduction to this system, this description provides enough information to read the sections dedicated to ACL2 topics. A complete description of ACL2 can be found in [KMM00b, KM].

In addition, an interesting ACL2 feature, that will be really important in our developments, is described in this section, too. Some other necessary concepts about ACL2 will be introduced later on for a better understanding of this memoir.
1.3 ACL2

1.3.1 Basics on ACL2

Information in this section has been mainly extracted from [KMM00b].

ACL2 stands for **A** _Computational Logic for A**pplicative **C**ommon **L**isp_. ACL2 is a programming language, a logic and a theorem prover. Thus, the system constitutes an environment in which algorithms can be defined and executed, and their properties can be formally specified with the assistance of a mechanical theorem prover.

The first version of ACL2 was developed by B. Boyer and J. S. Moore in 1989, using the functional programming language Common Lisp. ACL2 is the successor to the Nqthm [BM97] (or Boyer-Moore) logic and proof system and its Pc-Nqthm interactive enhancement. ACL2 was born as response to the problems Nqthm users faced in applying that system to large-scale proof projects. Namely, the main drawback of the Nqthm appears when the logic specification were used not only for reasoning about the modeling system but also for executing. The main difference between ACL2 and Nqthm lies in the chance of executing the models of the ACL2 logic in Common Lisp.

ACL2 and Nqthm have already shown their effectiveness for implementing (non-trivial) large proofs in mathematics. One of these examples can be found in [Kau00], where the Fundamental Theorem of Calculus is formalized; another development can be found in [BM84], where a mechanical proof of the unsolvability of the halting problem is implemented; other examples of theorems certified with ACL2 and Nqthm are: the Gauss’s Law of Quadratic Reciprocity [Rus92], the Church-Rosser theorem for lambda calculus [Sha88], the Gödel’s incompleteness theorem [Sha94], and so on.

Besides, ACL2 has been used for a variety of important formal methods projects of industrial and commercial interest, including (for example) the verification of the RTL code which implements elementary floating point operations of the AMD Athlon processor [Rus98] and ROM microcode programs of CAP digital signal processor of Motorola [BKM96].

More references about these and other developments related to ACL2 and Nqthm are www-available at [KM, BM97].

After this brief historical introduction, let us devote some lines to explain the programming language, the logic and the theorem prover of ACL2.

As a programming language, ACL2 is an extension of an applicative subset of Common Lisp. The logic considers every function defined in the programming language as a first-order function in the mathematical sense. For that reason, the programming language is restricted to the applicative subset of Common Lisp. This means, for example, that there is no side-effects, no global variables, no destructive updates and all the functions must be total and terminate. Even with these restrictions, there is a close connection between ACL2 and Common Lisp: ACL2 primitives that are also Common Lisp primitives behave exactly in the same way, and this means that, in general, ACL2 programs can be executed in any compliant Common Lisp.
The ACL2 logic is a first-order logic, in which formulas are written in *prefix notation*; they are quantifier free and the variables in them are implicitly universally quantified. The logic includes axioms for propositional logic (with connectives \texttt{implies}, \texttt{and}, \ldots), equality (\texttt{equal}) and properties related to a subset of primitive Common Lisp functions. Rules of inference include those for propositional logic, equality and instantiation of variables. The logic also provides a principle of proof by induction that allows the user to prove a conjecture splitting it into cases and inductively assuming some instances of the conjecture that are smaller with respect to some well founded measure.

An interesting feature of ACL2 is that the same language is used to define programs and to specify properties of those programs. Every time a function is defined with \texttt{defun}, in addition to define a program, it is also introduced as an axiom in the logic. Theorems and lemmas are stated in ACL2 by the \texttt{defthm} command, and this command also starts a proof attempt in the ACL2 theorem prover. In the ACL2 jargon a file containing definitions and statements that have been certified as admissible by the system is called a \textit{book}.

The main proof techniques used by ACL2 in a proof attempt are simplification and induction. The theorem prover is automatic in the sense that once \texttt{defthm} is invoked, the user can no longer interact with the system. However, in a deeper sense the system is interactive: very often non-trivial proofs are not found by the system in a first attempt and then it is needed to guide the prover by adding lemmas, suggested by a preconceived hand proof or by inspection of failed proofs. These lemmas are then used as rewrite rules in subsequent proof attempts. This kind of interaction with the system is called “The Method” by ACL2 authors.

In the following subsection, an introduction to an interesting ACL2 feature, that will be critical in our developments, is presented.

### 1.3.2 ACL2 encapsulates

In this subsection, an introduction to the ACL2 encapsulates is given. Encapsulates are a powerful tool which allow the structural development of theories. The encapsulation facility is much more general than sketched here, see [KMM00b, KMM00a, BGKM91, KM01]. They will be used, in this memoir, in Section 4.3 and onward.

The \textit{encapsulate principle} allows the introduction of function symbols in ACL2, without a complete specification of them, but just assuming some properties which partially define them. To be admissible and to preserve the consistency, there must exist some functions (called “local witness”) verifying the properties assumed as axioms. Once this is proved, the local witness definitions can be neglected, and the function symbols introduced with the encapsulate principle are partially specified by means of the assumed properties. Then, we can assume a property described by the formula \( \Phi \) for the functions \( f_1, \ldots, f_n \) if:
• \( f_1, \ldots, f_n \) are new function symbols.

• There exist admissible definitions of \( n \) functions whose names are \( f_1, \ldots, f_n \) such that the formula \( F \) can be proved for those functions.

If the encapsulate is admissible, the function symbols \( f_1, \ldots, f_n \) are added to the ACL2 language and the formula \( F \) is added as axiom to the logic. It is worth noting that the local witness are only used to ensure the admissibility of the encapsulate, since the axioms associated with their definitions are not kept.

In order to clarify this ACL2 mechanism an example will be presented. Let us suppose that we want to assume the existence of a binary function which is associative and commutative. The ACL2 command in charge of this task is the following one.

```lisp
(encapsulate
  ; the signatures
  (((op * *) => *))
  
  ; the witnesses
  (local (defun op (a b) (+ a b)))
  
  ; the axioms
  (defthm op-associative
    (equal (op (op x y) z) (op x (op y z))))

  (defthm op-commutative
    (equal (op x y) (op y x)))
)
```

The first expression is a list with the arity description of the functions which are introduced with the encapsulate principle (called signature). In this case, we indicate that \( \text{op} \) is a function with two arguments which returns a unique value. The local witnesses are defined using `defun`, but they are declared as local by means of `local`. The properties assumed over the encapsulate functions are stated by means of `defthm`. The encapsulate can contain definitions and properties declared as local; however, once the admissibility of the encapsulate has been proved, just the non local definitions and properties are added to the ACL2 logic.

The functions defined by means of an encapsulate cannot be executed, since its partial specification can be not enough to deduce the value for every input. However, there exists a great advantage from the verification point of view; due to the fact that we are formalizing and verifying a generic result which can be instantiated for concrete cases later on. This issue will be presented in Subsection 6.1.1.