ARTICLE IN PRESS

Journal of Symbolic Computation ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Journal of Symbolic Computation

journal homepage: www.elsevier.com/locate/jsc

fKenzo: A user interface for computations in Algebraic Topology☆

J. Heras [a], V. Pascual [a], J. Rubio [a,1], F. Sergeraert [b]

[a] *Departamento de Matemáticas y Computación, Universidad de La Rioja. Logroño, Spain*

[b] *Institut Fourier, Université Joseph Fourier. Grenoble, France*

ARTICLE INFO

Article history:
Received 7 September 2010
Accepted 21 January 2011
Available online xxxx

Keywords:
Symbolic computation systems
User interface
Constructive Algebraic Topology

ABSTRACT

fKenzo (= *f*riendly *Kenzo*) is a *graphical user interface* providing a user-friendly front-end for the *Kenzo* system, a Common Lisp program devoted to Algebraic Topology. The *fKenzo* system provides the user interface itself, an XML intermediary generator-translator and, finally the *Kenzo* kernel. We describe in this paper the main points of *fKenzo*, and we explain also the advantages and limitations of *fKenzo* with respect to *Kenzo* itself. The text is separated into two parts, trying to cover both the user and the developer perspectives.

1. Introduction

Algebraic Topology studies the topological spaces by algebraic means, in particular through *algebraic invariants*, such as homology and homotopy groups. For example two spaces having *different* (non-isomorphic) homology groups certainly have *different* homotopy types.[2] The modern evolution of Algebraic Topology stresses the *structural* problems, somewhat giving up its initial *computational* flavor. A possible reason of this evolution could be that calculating homology or homotopy groups of *arbitrary* spaces in general is a difficult task, outside the scope of a topologist working "only" with pen and paper.

The striking example of Commutative Algebra, where powerful computational tools such as Macaulay, Cocoa or Singular are commonly used for a long time by the specialists to help their

☆ Partially supported by MICIN, project MTM2009-13842-C02-01.

E-mail addresses: jonathan.heras@unirioja.es (J. Heras), vico.pascual@unirioja.es (V. Pascual), julio.rubio@unirioja.es (J. Rubio), francis.sergeraert@ujf-grenoble.fr (F. Sergeraert).

1 Tel.: +34 941299448; fax: +34 941299460.

2 The converse is unfortunately (or fortunately, depending on the point of view) false.

0747-7171/$ – see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jsc.2011.01.005

Please cite this article in press as: Heras, J., et al., *fKenzo*: A user interface for computations in Algebraic Topology. Journal of Symbolic Computation (2011), doi:10.1016/j.jsc.2011.01.005

work, frequently leading to new fascinating *mathematical* problems, has no counterpart in Algebraic Topology. Why? An interesting subject for historians of mathematics. A possible explanation is the following: Algebraic Topology programs require a high level of *functional programming*, but it is not the subject of this paper.

This paper is devoted to a presentation of the *fKenzo* program, a user-friendly front-end allowing a topologist to use the *Kenzo* program for simple applications, without being disconcerted by the Lisp technicalities which are unavoidable when using all the possibilities of *Kenzo*.

To illustrate the computing abilities of *fKenzo*, let us consider a hypothetical scenario where a graduate course is devoted to *fibrations*, in particular introducing the functors *loop space* Ω and *classifying space* B. In the simplicial framework, May (1967) is a good reference for these subjects. In this framework, if X is a connected *space*, its loop space ΩX is a simplicial *group*, the structural group of a universal fibration $\Omega X \hookrightarrow PX \to X$. Conversely, if G is a simplicial *group*, the classifying space BG is also the base space of a universal fibration $G \hookrightarrow EG \to BG$. The obvious symmetry between both situations naturally leads to the question: in an appropriate context, are the functors Ω and B inverse of each other? For the composition $B\Omega$, using *fKenzo* to compare for example the first homology groups of S^2, ΩS^2 and $B\Omega S^2$ gives some plausibility to the relation $B\Omega = \mathrm{id}$ in the homotopy category. The simplicial group ΩS^2 can in turn be used to compare in the same way ΩS^2 and $\Omega B\Omega S^2$ with the same conclusion.

A (good) student could wonder why the simpler case of S^1 has not been considered. Using again *fKenzo* this time fails; yet the result $B\Omega S^1 \sim S^1$ is true. But the Eilenberg–Moore spectral sequence cannot be used in this case to compute $H_*\Omega S^1$, for S^1 is not simply connected, and *fKenzo* checks this point. The symmetric comparison between S^1 and ΩBS^1 fails too, for another reason: the "standard" S^1 is a topological group, but the standard simplicial presentation of S^1 cannot be endowed with a structure of *simplicial* group. This is a good opportunity to introduce the Eilenberg–MacLane space $K(\mathbb{Z}, 1)$, the "minimal" Kan model of the circle S^1, a simplicial group; and the *fKenzo* comparison between the first homology groups of $K(\mathbb{Z}, 1)$ and $\Omega BK(\mathbb{Z}, 1)$ does give the expected result.

We think that this illustrates how *fKenzo* can be used as a research tool, precisely a specialized computer tool, for Algebraic Topology. It is worth noting that all the spaces in the examples (except the spheres S^1 and S^2) are not simplicial sets of finite type. Thus computing their homology groups, without the appropriate tool, is a challenging task, beyond the capabilities of beginners in Algebraic Topology. Fortunately, the program *Kenzo* can quickly calculate these groups, giving to Algebraic Topology an experimental feature, as Commutative Algebra inherited many years ago from Computer Algebra systems.

The paper is organized in two parts. In the first one (Section 2), the user point of view is stressed. By using the examples of this introduction about loop spaces and classifying spaces, the respective work styles in *Kenzo* (Section 2.1) and *fKenzo* (2.2.1) are illustrated. A short overview on *fKenzo* capabilities is the subject of Section 2.2.2. In the last section of this first part, we compare the advantages and drawbacks of *Kenzo* vs *fKenzo*.

In the second part some explanations on the development of the *fKenzo* system are given. We hope our experiences can be useful for other researchers undertaking similar tasks. Section 3.1 is devoted to architectural and technological issues. The user interaction design is described in Section 3.2. Finally, some challenges we faced and our proposed solutions are presented in Section 3.3.

The paper ends with a section dealing with conclusions and future work section, and the bibliography.

2. *Kenzo* and *fKenzo*

2.1. *Kenzo* before *fKenzo*

The original *Kenzo* program is a Common Lisp package, to be used in a Common Lisp environment. The *Kenzo* web page (Dousson et al., 1999) gives the relevant information allowing a topologist to

Please cite this article in press as: Heras, J., et al., *fKenzo*: A user interface for computations in Algebraic Topology. Journal of Symbolic Computation (2011), doi:10.1016/j.jsc.2011.01.005

install a Common Lisp environment on his laptop,[3] and then to install *Kenzo*; this program is nothing but a bunch of additional *classes* (chain complexes, simplicial sets, simplicial groups, …) and a large number of appropriate functions allowing the user to handle the traditional objects of Algebraic Topology, in particular to compute many homology and homotopy groups.

A *Kenzo* session trying to compare the 2-sphere S^2 and the classifying space $B\Omega S^2$ could start as follows:

```
> (setf S2 (sphere 2)) ✠
[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 S2 (sphere 2))` 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. Here the 2-sphere S^2 is constructed by the *Kenzo* function `sphere`, taking account of the argument 2, and this sphere is *assigned* to the Lisp symbol `S2` for later use. Also evaluating a Lisp statement *returns* an object, the result of the evaluation, in this case the Lisp object implementing the 2-sphere, 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 2-sphere, in our running example). In addition, another function allows the user to obtain the *origin* of the object (i.e. from which function and with which arguments it 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 loop space ΩS^2, a simplicial *group*, and in turn the classifying space of this group $B\Omega S^2$, which is only a simplicial *set*, since the group ΩS^2 is not abelian.

```
> (setf OS2 (loop-space S2)) ✠
[K6 Simplicial-Group]
> (setf BOS2 (classifying-space OS2)) ✠
[K18 Simplicial-Set]
```

It is well known that $H_p\Omega S^2 = \mathbb{Z}$ for every $p \geq 0$, which can be directly computed by *Kenzo*. To this aim, *Kenzo*, using a simple and powerful algorithm (see Rubio and Sergeraert, 1988; Sergeraert, 1994 or Berciano et al., 2010) can compute $H_*\Omega^k X$ if X is a k-connected simplicial set with effective homology. For the particular case $k = 1$ it is nothing but the Adams Cobar construction. In our concrete example:

```
> (homology OS2 6) ✠
Homology in dimension 6 :
Component Z
```

to be interpreted as stating $H_6\Omega S^2 = \mathbb{Z}$. We can therefore compare the homology groups of S^2 and $B\Omega S^2$.

```
> (homology S2 2) ✠
Homology in dimension 2 :
Component Z
> (homology BOS2 2) ✠
Homology in dimension 2 :
Component Z
> (homology S2 6) ✠
Homology in dimension 6 :

> (homology BOS2 6) ✠
Homology in dimension 6 :
```

[3] Some Common Lisp environments are free.

Please cite this article in press as: Heras, J., et al., *fKenzo*: A user interface for computations in Algebraic Topology. Journal of Symbolic Computation (2011), doi:10.1016/j.jsc.2011.01.005

No *component* displayed after a title as `Homology in dimension 6` means the corresponding homology group is null. *Kenzo* so informs the user $H_pS^2 = H_pB\Omega S^2$ for $p = 2$ and $p = 6$ and other analogous experiences for other values of p give the same result, which implies the question of a homotopy equivalence $S^2 \overset{?}{\sim} B\Omega S^2$ deserves to be studied.[4]

The symmetric question $G \overset{?}{\sim} \Omega BG$ requires a *group* G, for example $G = \Omega S^2$. The reader can now understand the goal of the next *Kenzo* statements.

```
> (setf OBOS2 (loop-space BOS2)) ✠
[K252 Simplicial-Group]
> (homology OBOS2 6) ✠
Homology in dimension 6 :
Component Z
```

and again the user can check $H_p\Omega S^2 = H_p\Omega B\Omega S^2$ for small values of p. The same calculations can be repeated with other initial spaces, clearly suggesting the general question of a homotopy equivalence $G \overset{?}{\sim} \Omega BG$.

Let us do the same work for the multiplicative group S^1 of the complex numbers of modulus 1. In particular the classifying space BS^1 is defined. The group S^1 is also a 1-dimensional *sphere*, and *Kenzo* can construct spheres:

```
> (setf S1 (sphere 1)) ✠
[K395 Simplicial-Set]
> (setf BS1 (classifying-space S1)) ✠
Error: No methods applicable for generic function
       #<STANDARD-GENERIC-FUNCTION CLASSIFYING-SPACE>
       with args ([K395 Simplicial-Set]) of classes (SIMPLICIAL-SET)
```

But the default implementation of the circle S^1 is the simplicial *set* with one vertex and one edge. As clearly displayed, this object has a structure (*class* in computer jargon) of simplicial *set*, so that the generic function `classifying-space` cannot be applied to such an object. Computational algebraic topology must use *combinatorial* models and the simple simplicial model of the circle cannot be endowed with a structure of a simplicial *group*: a simplicial group is necessarily a Kan simplicial set, see May (1967), and the minimal Kan model of the circle is the standard simplicial model of the Eilenberg–MacLane space $K(\mathbb{Z}, 1)$; it is a simplicial set *not of finite type* but which nevertheless can be constructed and used under *Kenzo*.

```
> (setf KZ1 (k-z 1)) ✠
[K513 Abelian-Simplicial-Group]
> (setf BKZ1 (classifying-space kz1)) ✠
[K525 Abelian-Simplicial-Group]
> (setf OBKZ1 (loop-space bkz1)) ✠
[K537 Simplicial-Group]
```

The homology groups $H_p\Omega BK(\mathbb{Z}, 1)$ can then be computed and compared with the well-known homology groups of the circle.

```
> (homology OBKZ1 1) ✠
Homology in dimension 1 :
Component Z
> (homology OBKZ1 6) ✠
Homology in dimension 6 :
```

[4] *Kenzo* allows also the user for example to ask for *every* homology group $H_pB\Omega S^2$ with $p \leq 8$, more conveniently, but leads to verbose output to be avoided in this short note.

Please cite this article in press as: Heras, J., et al., *fKenzo*: A user interface for computations in Algebraic Topology. Journal of Symbolic Computation (2011), doi:10.1016/j.jsc.2011.01.005

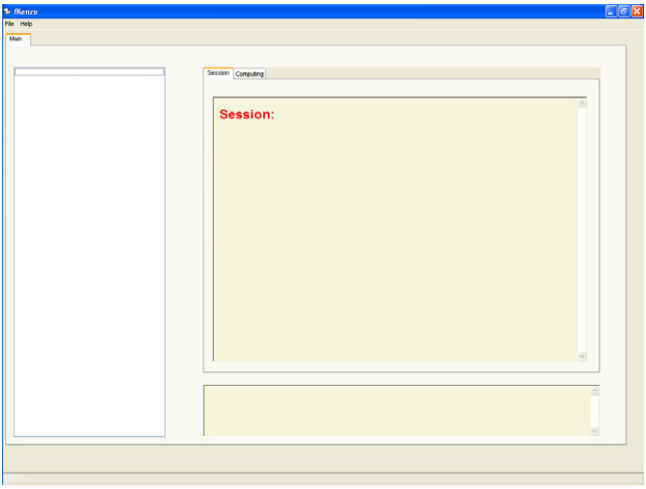

Fig. 1. Initial screen of *fKenzo*.

2.2. *fKenzo in action*

2.2.1. *The same example with fKenzo*

To repeat the same computations with *fKenzo*, one can go to Heras (2009) and download the installer. After the installation process (no Common Lisp independent installation is needed), you can click on the *fKenzo* icon, accessing an initial, "empty" interface, shown in Fig. 1. This initial interface contains only two menus: *File* and *Help*. In the *Help* menu the beginner can find some documentation explaining how he can start working with *fKenzo*. In particular, a description of the first steps and the functionality included in each module can be found there.

Since we are planning to work with simplicial sets and simplicial groups, we use *File* → *Add Module*, and then we choose `Simplicial-Groups.omdoc`. The interface is updated with a new menu called *Simplicial Sets* and another one called *Simplicial Groups*. When deploying them, we find several options, to construct in particular spheres, loop-spaces or classifying spaces. When selecting the "sphere" option in the *Simplicial Set* menu, *fKenzo* asks for a natural number limited, by default in *Kenzo*, to 14. Then (see Fig. 2) the space denoted by `SS 1` appears in the left side of the screen; when selecting it, the mathematical notation of the space appears in the bottom part of the right side of the panel. The history of the constructed spaces is shown in the session tab, top part of the right side of the panel.

If we try to construct a classifying space from the *Simplicial Group* menu, *fKenzo* informs us that it needs a simplicial group (thus likely an error is avoided). We can then construct the space ΩS^2. Since ΩS^2 is the only simplicial group in this session, when selecting the classifying space option, ΩS^2 is the only available space appearing in the list which *fKenzo* shows. In order to work with Eilenberg–MacLane spaces, the *Abelian Simplicial Group* module should also be loaded, in this way all the spaces used in the previous section can be built, as can be seen in Fig. 2.

Loading the `Computing.omdoc` file, the menu *Computing* where we can select "homology" becomes available. In this manner, the computations performed in Section 2.1 can be reproduced, as can be seen in Fig. 3.

If the user tries a calculation with is not supported by *Kenzo*, as in the case of $H_4(B\Omega^2 S^2)$, *fKenzo* informs him that $H_4(B\Omega^2 S^2)$ is not computable (see Fig. 4) due in this particular situation to a problem with the connectivity of the space. Thus, *fKenzo* not only computes in a more friendly way than *Kenzo*, but also leads the user, avoiding running errors (see the idea of intermediary layer in Section 3.1 and a more detailed description in Heras et al. (2008)).

Please cite this article in press as: Heras, J., et al., *fKenzo*: A user interface for computations in Algebraic Topology. Journal of Symbolic Computation (2011), doi:10.1016/j.jsc.2011.01.005

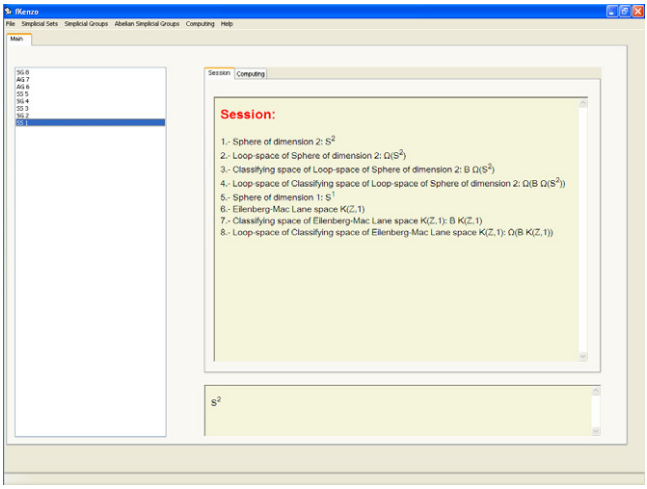

Fig. 2. Example of session.

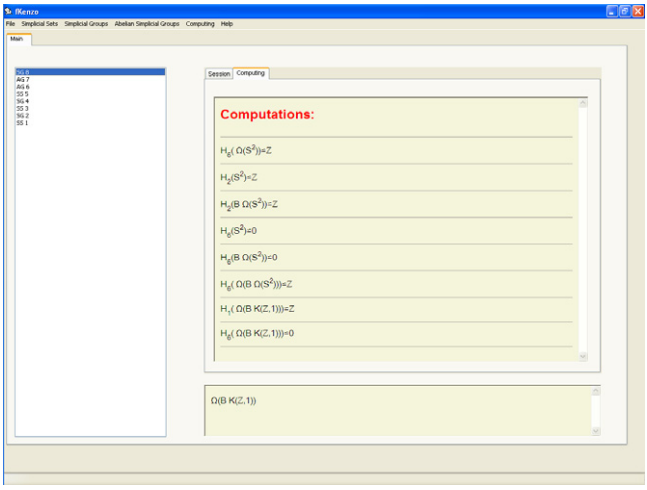

Fig. 3. Example of computations.

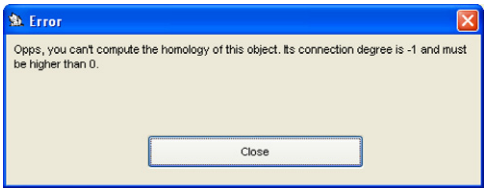

Fig. 4. Reduction degree error.

2.2.2. *fKenzo: system overview*

In its current distribution, *fKenzo* contains five modules: *Chain Complexes*, *Simplicial Sets*, *Simplicial Groups*, *Abelian Simplicial Groups* and *Computing*. In addition, some other *experimental* modules can be

Please cite this article in press as: Heras, J., et al., *fKenzo*: A user interface for computations in Algebraic Topology. Journal of Symbolic Computation (2011), doi:10.1016/j.jsc.2011.01.005

downloaded, through the plug-in manager of the Help menu, such as a possibility of interfacing with the GAP Computer Algebra system or the ACL2 Theorem Proving tool.[5]

The *Simplicial Set* module contains most of the functionality of *Kenzo*: options to construct spaces from scratch (spheres, Moore spaces, finite simplicial sets, and so on) and from other spaces (cartesian products, suspensions, wedges, and so on). *Simplicial Groups* module contains the operations to build loop spaces and classifying spaces, in addition to an automatic loading of the *Simplicial Set* module which is needed to build loop spaces. *Abelian Simplicial Groups* module contains just one option (Eilenberg–MacLane spaces). The *Chain Complexes* module includes a few operations that are defined at the algebraic level but not at the simplicial one, as tensor products.

Besides these structural modules, there is a special module *Computing*,[6] allowing the user to calculate homology and homotopy groups of the constructed spaces. It is necessary to understand that homology and homotopy algorithms are very different. From its very construction (as objects with effective homology, see Rubio and Sergeraert (2002)), the (first) homology groups of each space constructed with *Kenzo*/*fKenzo* are usually available. On the contrary, *fKenzo* cannot compute homotopy groups of *all* the spaces which can be constructed with the system (see the documentation of *Kenzo* or *fKenzo* to know more about this topic).

With respect to the visual aspect of the *fKenzo* panel, it has been briefly described in the previous subsection. The "Main" tab contains, at its left side, a list of spaces constructed in the current session, identified by its type (CC = Chain Complex, SS = Simplicial Set, SG = Simplicial Group, AG = Abelian Simplicial Group) and its internal identification number (this identification number is also used internally to keep track of the origin of spaces, allowing the program to reuse intermediate results, getting a performance comparable to that of *Kenzo*, which also uses this technique. When selecting one of the spaces in this list, its standard notation appears at the bottom part of the right side. At the upper part, there are two tabs: "Session" (containing a textual description of the constructions made, see Fig. 2; this can be saved, rendered in external browsers and recovered for further working, if it is wanted by the user) and "Computing" (containing the homology and homotopy groups computed in the session; these results can also be saved and rendered in external browsers, but they cannot be reloaded into *fKenzo*, since they are not used in further computations).

2.3. Kenzo beyond fKenzo

Algebraic Topology is a vast and complex subject, in particular mixing Algebra and (combinatorial) Topology. The program designers in Symbolic Computation always meet the same decision problem; two possible organizations[7]:

(1) Provide a package of procedures in the programming language L, allowing a user of this language to load this package in the standard L-environment, and to use the various functions and procedures provided in this package. It is in particular the solution followed in *Kenzo* with respect to the Common Lisp language. Advantage: the total freedom given by the language L remains available; Disadvantage: the technicalities of the language L remain present as well!
(2) Provide a graphical interface with the usual tabs, menus and other widgets. It is the *fKenzo* style. Advantage: to give to an inexperienced user a direct access to the most simple desired calculations, without having to learn the language L; Disadvantage: some functionalities could be difficult to fit in the Graphical User Interface, while easily programmable with the language L.

[5] These experimental aspects are the main reason why we use the XML standard OpenMath (Buswell et al., 2004) as specification language: it allows us to communicate with other Computer Algebra Systems, as GAP, and the axiomatic information in OpenMath Content Dictionaries can be exploited to prove some properties of programs with ACL2. More details on these technological issues are given in the second part of this paper (Section 3).

[6] Even if *fKenzo* is always used for "computing",the process needs two very different steps. In Step 1 (Constructing), the asked-for spaces and many auxiliary other objects are constructed, each one being essentially a small set of *functional objects*, ready to answer various questions. In Step 2 (Computing), taking account of the demanded group, *Kenzo* does question most of these functional objects and, combining in a rather sophisticated way the obtained answers, finally returns the wished answer.

[7] These are, of course, two extreme positions: many other possibilities can be explored between them.

Please cite this article in press as: Heras, J., et al., *fKenzo*: A user interface for computations in Algebraic Topology. Journal of Symbolic Computation (2011), doi:10.1016/j.jsc.2011.01.005

We give two examples of results that are reachable in the original *Kenzo* environment, which on the contrary are beyond the scope of *fKenzo* and seem difficult to introduce in any other sensible graphical interface.

In the paper Sergeraert (2010), the following game is tried, and rather amazingly succeeds. Let X be the complex projective space $P^n\mathbb{C}$. It is a subset of the infinite complex projective space $P^\infty\mathbb{C}$, and the fundamental class of $P^n\mathbb{C}$ is also the generator of $H_{2n}P^\infty\mathbb{C}$. It happens that $P^\infty\mathbb{C}$ has the homotopy type of the Eilenberg–MacLane space $K(\mathbb{Z}, 2)$. This Eilenberg–MacLane space has a canonical *minimal* version as a simplicial set K_2, an important object when computing homotopy groups in *Kenzo*. The simplicial set K_2 can be constructed in *Kenzo*, and in *fKenzo* as well. This simplicial set is "minimal" but not at all of finite type; yet the methods of *effective homology* allow us to compute the effective homology of K_2. In particular *Kenzo* can produce a representative c, a cycle, of a generator of $H_{2n}(K_2)$. The cycle c is a finite $\mathbb{Z}$-combination of $2n$-simplices of K_2. Then *Kenzo* can construct the sub-simplicial set $S_c \subset K_2$ generated by the components of c. It is not then very hard to prove that if the relative homology $H_*(K_2, S_c)$ is null up to dimension $2n + 1$, then the simplicial set S_c has the homotopy type of $P^n\mathbb{C}$. We can use *Kenzo* to compute this relative homology, obtaining the desired vanishing property. This proves that S_c is a triangulation as a simplicial *set* of (the homotopy type of) $P^n\mathbb{C}$. We thus obtain in a few seconds a triangulation of (the homotopy type of) $P^5\mathbb{C}$ with (1, 0, 5, 40, 271, 1197, 3381, 5985, 6405, 3780, 945) simplices, that is, 1 vertex, 5 triangles, 40 tetrahedrons, …, 945 10-simplices. In particular, a compact triangulation of $P^2\mathbb{C}$ as a simplicial set with (1, 0, 2, 3, 3) simplices is obtained, to our knowledge not yet known.

The reader can understand that this set of calculations requires a complex user interaction, in particular invoking explicit $\mathbb{Z}$-cycles. In *fKenzo*, every object such as a chain complex, a simplicial set, …, is implemented in a *global* way as a *finite* set of *functional* objects; such a functional object *will later* work on arguments most often quite complex, using all the technicalities of the underlying Common Lisp language; it is difficult to find a way of giving the user of a Graphical User Interface access to such arguments: working directly in a Common Lisp environment is much easier; so that if our user wants to define and handle himself such arguments, using *Kenzo* itself seems then a good alternative.

Another example is the subject of the paper (Berciano et al., 2010). An A_∞-structure on a chain complex C_* is a multiplication μ_2 defined over C_*, compatible with the differential, but associative only up to homotopy; such an explicit homotopy $\mu_3 : C_* \otimes C_* \otimes C_* \to C_*$ must be provided, which in turn in a sense must verify some associativity property up to an *explicit* homotopy μ_4 and so on. See Kadeishvili (2008) for a convenient description of this relatively complex structure, discovered by Jim Stasheff in the sixties; see Stasheff (1963). Such an A_∞-structure is complex but can be easily constructed by *Kenzo* in some contexts, as a consequence of the powerful Basic Perturbation Lemma. The paper Berciano et al. (2010) explains how highly non-trivial A_∞-structures can be constructed by *Kenzo*, when applying again the methods of effective homology to the homology groups $H_*(\Omega^3(P^\infty\mathbb{R}/P^3\mathbb{R}))$. Some intermediate chain complexes which are necessary to compute these homology groups are endowed with an A_∞-structure $(\mu_n)_{n\geq 1}$, and a careful analysis of this structure shows every μ_n is non-trivial. This requires a study of terms $\mu_n(g \otimes \cdots \otimes g)$, and such a meticulous study is difficult to integrate with the general *fKenzo* style, for the same reasons as in the previous example.

3. *fKenzo*: system description

3.1. Methodological and technological issues

3.1.1. Architecture of the system

The general organization of *fKenzo* has been inspired by the Microkernel pattern (see Buschmann et al. (1996)). The main component of this pattern, called *mediator* in our framework, is responsible for managing all system resources, maintains information about resources and allows access to them in a coordinated and systematic way. A high level perspective of the system as a whole is shown in Fig. 5.

Please cite this article in press as: Heras, J., et al., *fKenzo*: A user interface for computations in Algebraic Topology. Journal of Symbolic Computation (2011), doi:10.1016/j.jsc.2011.01.005

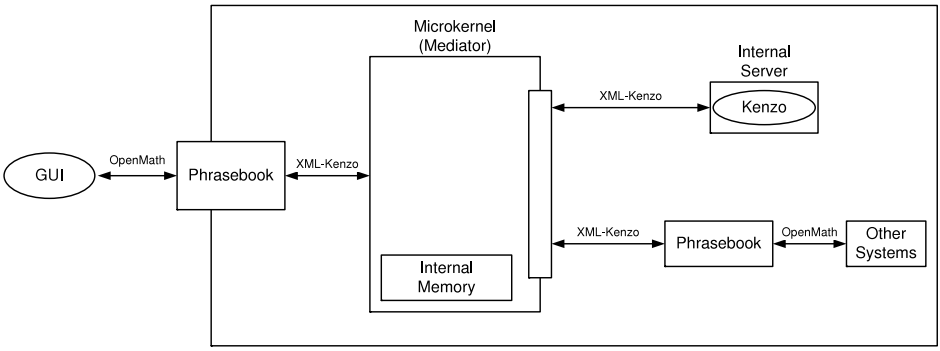

Fig. 5. *fKenzo* Microkernel architecture.

In *fKenzo*, the Microkernel pattern is specialized in two different ways. First, our mediator embeds an *Internal Memory*, which acts as a working memory when running a process, handling objects by means of identification numbers (visible at the Graphical User Interface, as explained in Section 2.2), and improving the performance of the system. Second, the communications from and to the mediator are encoded by means of an XML language (called *XML-Kenzo*) that frees us from the implementation languages occurring in the different components of the system. These issues are dealt with in the following subsection. Remark in Fig. 5 that the main computing kernel is *Kenzo* itself, wrapped with an XML processor. Other computing or deduction engines can be also integrated. In Fig. 5, these other kernels have been linked by means of OpenMath processors (*Phrasebooks* in OpenMath terminology; see Buswell et al. (2004)), because it is the technology used in our current experimental prototypes (to GAP and ACL2, as invoked previously).

3.1.2. Modularity and OpenMath

Modularity has two aims in *fKenzo*. One of them is related to the separation of concerns in the user interface. The second one allows us to design a dynamically extensible system, where modules are plugged in. In both cases, OpenMath technology (OMDoc documents, concretely) is instrumental to implement these ideas.

With respect to the first aspect, our inspiration comes from Hanus and Kluß (2009), where a proposal for the declarative programming of user interfaces was presented. In Hanus and Kluß (2009), the authors distinguished three constituents in any user interface: structure, functionality and layout (this separation of concerns is very related to the well-known *Model-View-Controller* pattern; see Buschmann et al. (1996) for details). These three parts are encoded for each module of *fKenzo* in an OMDoc document (see Kohlhase (2006)). OMDoc is an open markup language for mathematical documents, and the knowledge encapsulated in them. This format extends OpenMath and hence provides some features not available in OpenMath, for example a theory level and a way of incorporating executable code. These enhancements are used in *fKenzo* to integrate in a unique OMDoc file for each module the different declarative parts of the interface: structure (encoded in XUL, the Mozilla's XML user interface language; see Hyatt et al. (2001)), functionality (Allegro Common Lisp programs) and layout (style sheets).

This organization also allows us to deal with the second modularity aspect. Since each user interface unit is encoded in a unique OMDoc file (with its inner modular organization: structure, functionality, layout), our front-end becomes *extensible*. It is enough to produce an OMDoc file with the suitable structure, and then it can be interpreted and plugged in our Graphical User Interface. It is exactly what happened when in Section 2.2.1 we described the way of working with *fKenzo*: the option *Add Module* with the Simplicial-Groups.omdoc file interprets indeed the OMDoc importing new structure, functionality and layout into the GUI. This extensibility principle makes very easy to us to incorporate experimental features to the system, without interfering with the already running modules.

Please cite this article in press as: Heras, J., et al., *fKenzo*: A user interface for computations in Algebraic Topology. Journal of Symbolic Computation (2011), doi:10.1016/j.jsc.2011.01.005

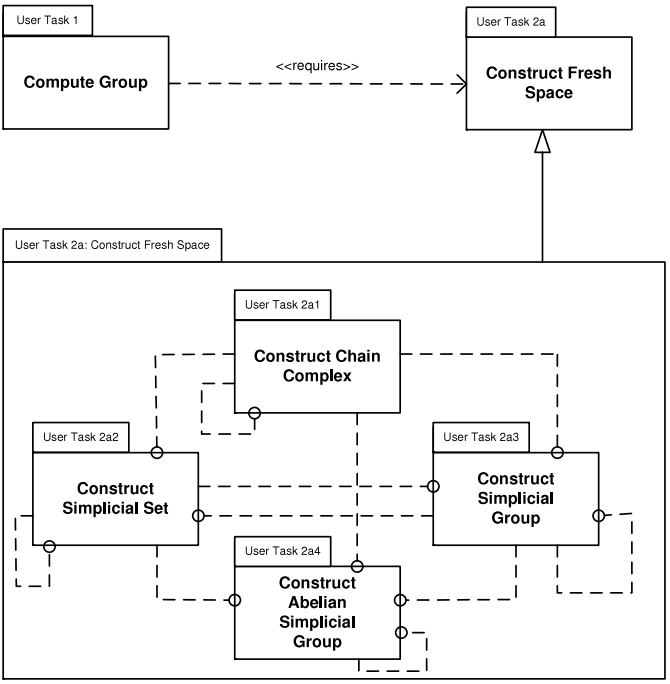

Fig. 6. Hierarchical decomposition of the "Construct Fresh Space" user task.

For a more detailed account, we refer to Heras et al. (2009) where we reported on this kind of interface organization.

3.2. Interaction design

3.2.1. Task model

The first idea guiding the construction of a user interface must be the objectives of the interaction. In *fKenzo* there is only one higher-level objective: to compute homology groups of spaces. This main objective is later on broken in several subobjectives, trying to emulate the way of thinking of a typical *Kenzo* user. Once this first objective analysis is done, the next step is to design a *task model*. That is to say, a hierarchical planning of the main actions the user should undertake to get his objectives. This is a previous step before devising the navigation of the user, which will give the concrete guidelines needed to implement the interface.

In our case, the two main tasks of the system are: (1) computing groups, and (2) constructing spaces. Note that the second task is necessary to realize the first one. In turn, the task of constructing spaces can be separated into: (1) constructing new fresh spaces and (2) loading spaces from a previous session. Thus, the notion of *session* comes on the scene. With respect to the construction of fresh spaces, once the user has decided to go for it, he should decide which type of space he wants to build: simplicial set, simplicial group, and so on. This third layer of tasks gives us the *module* organization of the interface, while *computing* produces a separated module, and the handling of *sessions*, being conceptually different, does not give rise to a module.

The task design is organized hierarchically by diagrammatic means. See in Fig. 6 a first decomposition layer. Each task (each frame) is linked to auxiliary tasks (giving a horizontal dependency structure). Then, each frame is described in more detail (vertical structure) by making explicit its subtasks graph.

Please cite this article in press as: Heras, J., et al., *fKenzo*: A user interface for computations in Algebraic Topology. Journal of Symbolic Computation (2011), doi:10.1016/j.jsc.2011.01.005

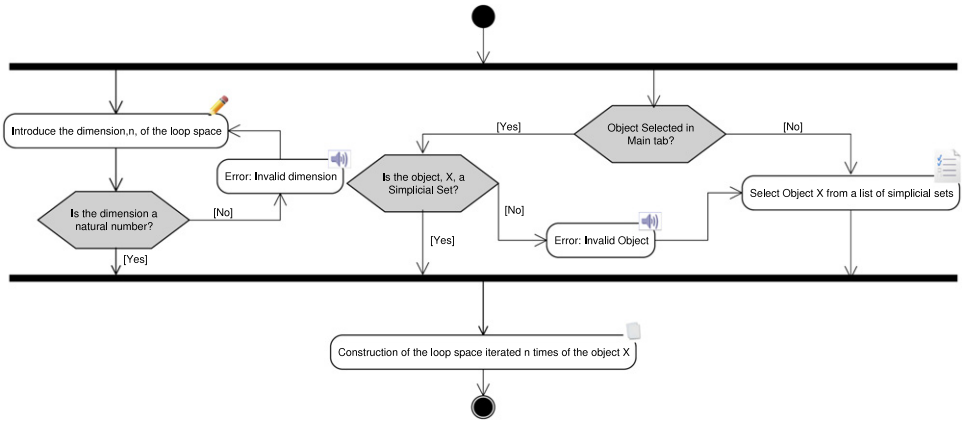

Fig. 7. Control graph for the construction of $\Omega^n X$.

Task modeling provides us with the high level modular structure and with the different steps needed to reach a user's subobjective. The concrete actions a user should perform to accomplish the tasks are devised in control and navigation models.

3.2.2. Control and navigation model

The design of the interaction between a user and a computer program involves well-known challenges (use of convenient metaphors, consistency of the control through the whole application, and so on). In order to avoid some frequent drawbacks we have followed the guidelines of the Noesis method (see Domínguez and Zapata (2007) for the general theory, and Cordero et al., 2006 for the design of reactive systems). In particular, our development has been supported by the Noesis models for control and navigation in user interfaces (as we reported in Heras et al. (2008)). These graph-supported models enable an exhaustive traversal of the interfaces, allowing the analyst to detect errors, disconnected areas, lack of uniformity, etc. *before* the programming phase. Fig. 7 shows the control and navigation submodel describing the construction of a loop space $\Omega^n X$. Different kinds of interactions are graphically represented in Fig. 7 by different icons. For instance, selecting from a list is depicted with a form icon; directly writing an input is depicted with a pen, and so on. These pictures help the programmer to get a quick overall view of the different controls to be implemented.

Let us observe that this diagrammatic control model is *abstract*, in the sense that nothing is said about the concrete way the transitions should be translated into the user interface. In fact, in *fKenzo* this model is implemented in two different manners: one by means of the "menu & mouse" style, and the other one through control-keys. The second style has been included thinking of advanced users, who want to use shortcuts to access the facilities of the interface. The adaptation to different kinds of users is one of the principles for design usability in Schneiderman (1998), and has been considered, as the rest of principles, in our development.

3.3. Challenges and design decisions

When starting the project of developing a user interface for *Kenzo*, several requirements were determined. Some of them were simply natural specifications, others were of a more problematic style. Those difficult issues are presented here as challenges to be fulfilled. These challenges largely determined the design decisions presented previously in this section. The links among ones (challenges) and others (design decisions) are explained in this subsection.

The most important challenges we faced were:

(1) *Extensibility.* The system design should be capable of evolving at the same time as the kernel *Kenzo* system. In addition, the system should be designed in such a way that it could support

Please cite this article in press as: Heras, J., et al., *fKenzo*: A user interface for computations in Algebraic Topology. Journal of Symbolic Computation (2011), doi:10.1016/j.jsc.2011.01.005

both other ways of interaction (web services, for example) and the connection to other symbolic manipulation systems (GAP in computational algebra, for instance, or ACL2 from the theorem proving side).

(2) *Efficiency*. The user interface should be roughly equivalent to *Kenzo* in time and space efficiency.

(3) *Error handling. fKenzo* should forbid the user some manipulations raising errors, both from structural and from semantical points of view.

(4) *Consistent metaphors.* An advanced user of *Kenzo* should feel comfortable with *fKenzo*; in particular, the typical two step process (first constructing a space, then computing groups associated to it) should be explicitly and graphically captured in *fKenzo*.[8]

(5) *Support to long term computations.* An important calculation in Algebraic Topology sometimes needs several days of processing; *fKenzo* should give the user the possibility of keeping a copy of his current session, that could be re-taken later on. Additionally, the session specifications should not be internal to the system, but it should be possible to export them and to communicate them to other users or computers.

Let us observe that, as it is frequent in system design, some decisions aimed to fulfill a concrete requirement could compromise other requirements. The most important trade-off in our previous list is between requirements (1) and (2). A layered architecture with complex mediators could produce poorer performance. A careless treatment of intermediary documents and files could also imply a great memory waste. Error handling (item (3)) could be in a conflict with efficiency, too, because dealing with semantical information at the external layers of an architecture can slow down the system as a whole. We have tried to deal with all these constraints while respecting the requirements.

To solve the first problem (extensibility) we use two well-known tools: design patterns to find an architectural solution and XML to manage the mathematical and systemic knowledge. As explained in Section 3.1.1, the *Microkernel* architectural pattern (see Buschmann et al. (1996)) was chosen to organize the system. The pattern was enhanced with XML processing capabilities both for the mathematical knowledge (XML-Kenzo and OpenMath) and for the very graphical user interface structure (through OMDocs files). We obtained a system that can incorporate new computing and deduction engines (due to the Microkernel platform organization) and whose user interface can be extended by simply loading OMDoc documents.

The second aspect (efficiency) is solved through *memoization*, a strategy also used by F. Sergeraert in *Kenzo*. The intermediary layer is in charge of keeping an enriched copy of the objects created, in such a way that re-calculations are avoided. To implement memoization, we needed to improve the Microkernel pattern with an "internal memory". As a result, the waiting time is to a great extent similar to that of the original *Kenzo* system.

The most important design decision related to point (3), error handling, is the addition of an "intelligent" processing in the mediator. To be more concrete, some structural constraints are directly wired in the graphical user interface; for instance, arguments for the constructor "classifying space" can only be selected among the spaces which are simplicial groups. This is accomplished because the OpenMath specification of spaces gives us a kind of (semantical) *type system* for spaces and objects in our working memory.

On the other hand, other information about the correct input requirements for computing homology groups (such as the *reduction degree*) is more dynamical in nature, and cannot be wired in the user's interface. This is dealt with in the mediator layer, symbolically manipulating the (XML) representation of spaces. The organization is as a small rule-based system (see Heras et al. (2008)), and the computational price is negligible with respect to ordinary computations in Algebraic Topology.

These "intelligent" enhancements have been got by using a *Decorator* pattern (see Buschmann et al. (1996)): every underlying *Kenzo* object is "decorated" in the intermediate *fKenzo* layer with an annotation of its reduction degree and other typing information. Then, when a computation

[8] This could be considered as an over-constraint to the design of the system; nevertheless, since the *fKenzo* designers were previously *Kenzo* users, it was natural for us to fix such a requirement.

Please cite this article in press as: Heras, J., et al., *fKenzo*: A user interface for computations in Algebraic Topology. Journal of Symbolic Computation (2011), doi:10.1016/j.jsc.2011.01.005

(of a homology group, for instance) is demanded by a user, the intermediary layer monitors if the annotation allows the transferring of the command to the *Kenzo* kernel, or if a warning must be sent to the user.

It is still controversial when some of these processes must be located at the external layer, in the intermediary layer or nearer to the *Kenzo* kernel. Up to now, we have followed the guideline of isolating *Kenzo* as much as possible. Since the performance overheads are reasonable this seems a good choice in the current state of *fKenzo*. In any case, our architecture is flexible enough to change the location of components (if scalability issues recommend it) with little impact on the rest of the system.

Requirement number (4), coherence with the *Kenzo* way of interaction, is the most influential with respect to the visual aspect of our interface. In addition to the menu bar, there are three main parts in the screen: a left part, with a listing of the objects already constructed in the current session, a right panel with several tabs, and a bottom part with the standard mathematical representation of the object selected (see Figs. 2 and 3). Thus, focus concentrates on the object (space) of interest, as in Common Lisp/*Kenzo*. The central panel takes up most of the place in the interface, because it is the most growing part of it (in particular, with respect to computing results). It is separated by means of tabs not only because on the division among spaces (Fig. 2) and (Fig. 3) computing results (the system moves from one to the other dynamically, putting the focus on the last user action), but also due to requirement number (1): since our interface should be capable of evolving to integrate other systems (computational algebra or theorem proving tools), playing with tabs in the central panel allows us to produce a user sensation of indefinite space and separation of concerns.

Finally, the fifth item has been the easiest to meet: once the XML infrastructure has been devised to fulfill requirements (1) and (3), it is easy to design a way of storing in a file the information about the spaces built during a session. This text files can be replayed by *fKenzo*, transmitted through nets or even displayed through a browser.

In summary, a good balance among requirements and design decisions seems to be achieved. There is much room for improvement, but a solid and stable first step has been given toward a usable interface for the *Kenzo* system.

4. Conclusions and further work

fKenzo is a user interface for the *Kenzo* system, a Common Lisp program to compute in Algebraic Topology. In its current state, *fKenzo* fulfills two objectives: it provides a friendly front-end to *Kenzo*, and it guides the user to avoid running errors, depending both on design decisions in *Kenzo* and on topological features. This second objective is got by means of a *mediator* program called *intermediary layer*. We hope these are right steps to reach our final aim of increasing the interest of algebraic topologists in *Kenzo*, or, more generally, in effective and constructive approaches to Algebraic Topology.

Three big lines of future work are open.

The first one is related to include more *Kenzo* functionalities in *fKenzo*. One of the aspects of this enhancement could be to find a suitable way (free from the Common Lisp syntax) of editing and handling elements of each constructed space. Thus we could approach the difficult question of introducing in *fKenzo* computations as the ones presented in Section 2.3.

The second line of work is related to the extensibility of *fKenzo*. Since the computational kernel, *Kenzo* itself, continues covering more aspects of Algebraic Topology (see for instance in Romero et al. (2006) an extension for spectral sequences), it is necessary that *fKenzo* evolves accordingly. The modular structure of *fKenzo*, based on OpenMath mechanisms, will be instrumental to this aim.

Finally, we would like *fKenzo* to become a comprehensive assistant for Algebraic Topology, including computation (the only aspect considered up to now), communication (with other systems) and deduction (by means of proof assistants). As evoked in the paper some preliminary developments on the connection with GAP and with the theorem prover ACL2 are already available through the current distribution of *fKenzo*.

Please cite this article in press as: Heras, J., et al., *fKenzo*: A user interface for computations in Algebraic Topology. Journal of Symbolic Computation (2011), doi:10.1016/j.jsc.2011.01.005

Acknowledgements

The authors would like to thank one of the anonymous referees for his valuable suggestions which significantly improved the paper.

References

Berciano, A., Rubio, J., Sergeraert, F., 2010. A case study of A_∞-structure. Georgian Mathematical Journal 17, 57–77.
Buschmann, F., Meunier, R., Rohnert, H., Sommerland, P., Stal, M., 1996. Pattern-oriented software architecture. In: A System of Patterns, vol. 1. Wiley.
Buswell, S., Caprotti, O., Carlisle, D.P., Dewar, M.C., Gaétano, M., Kohlhase, M., 2004. OpenMath Version 2.0. http://www.openmath.org/.
Cordero, C.C., Miguel, A.D., Domínguez, E., Zapata, M.A., 2006. Modelling interactive systems: an architecture guided by communication objects. In: HCI Related Papers of Interaccion 2004. Springer, pp. 345–357.
Domínguez, E., Zapata, M.A., 2007. Noesis: towards a situational method engineering technique. Information Systems 32 (2), 181–222.
Dousson, X., Sergeraert, F., Siret, Y., 1999. The Kenzo program. Institut Fourier, Grenoble, http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo/.
Hanus, M., Kluß, C., 2009. Declarative programming of user interfaces. In: Lecture Notes in Computer Science, vol. 5418. pp. 16–30.
Heras, J., 2009. *fKenzo*, http://www.unirioja.es/cu/joheras/.
Heras, J., Pascual, V., Rubio, J., 2008. Mediated acces to symbolic computation systems. In: Lecture Notes in Artificial Intelligence, vol. 5144. pp. 446–461.
Heras, J., Pascual, V., Rubio, J., 2009. Using open mathematical documents to interface computer algebra and proof assistant systems. In: Lecture Notes in Artificial Intelligence, vol. 5625. pp. 467–473.
Hyatt, D., Goodger, B., Hickson, I., Waterson, C., 2001. XML User Interface Language (XUL) 1.0. http://www.mozilla.org/projects/xul/.
Kadeishvili, T., 2008. Operadic algebraic topology. Ictp-Map 2008 Summer School. http://map.disi.unige.it/ictp/lectures_files/Kadeishvili_L.pdf.
Kohlhase, M., 2006. OMDoc — An open markup format for mathematical documents [Version 1.2]. Springer.
May, J.P., 1967. Simplicial Objects in Algebraic Topology. Van Nostrand.
Romero, A., Rubio, J., Sergeraert, F., 2006. Computing spectral sequences. Journal of Symbolic Computation 41 (10), 1059–1079.
Rubio, J., Sergeraert, F., 1988. Homologie effective et suites spectrales d'Eilenberg-Moore. Comptes Rendus de l'Académie Sciences Paris 306 (17), 723–726.
Rubio, J., Sergeraert, F., 2002. Constructive algebraic topology. Bulletin des Sciences Mathématiques 126 (5), 389–412.
Schneiderman, B., 1998. Designing the User Interface. Addison Wesley.
Sergeraert, F., 1994. The computability problem in algebraic topology. Advances in Mathematics 104, 1–29.
Sergeraert, F., 2010. Triangulations of complex projective spaces. In: Contribuciones Científicas en Honor de Mirian Andrés. pp. 507–519. http://www.unirioja.es/servicios/sp/catalogo/monografias/vr77.shtml.
Stasheff, J.D., 1963. Homotopy associativity of H-spaces, I, II. Transactions of the American Mathematical Society 108, 275–292. 293–312.

Please cite this article in press as: Heras, J., et al., *fKenzo*: A user interface for computations in Algebraic Topology. Journal of Symbolic Computation (2011), doi:10.1016/j.jsc.2011.01.005