Mechanising mathematics: the case of Algebraic Topology

Julio Rubio

Universidad de La Rioja
Departamento de Matemáticas y Computación

Tomás Recio 60

CIEM (Castro Urdiales), May 17th-21th, 2010

Joint work of the Programming and Symbolic Computation Team
https://esus.unirioja.es/psycotrip/

Partially supported by Ministerio de Educación y Ciencia, project MTM2009-13842-C02-01, and by European Commission FP7, STREP project ForMath.

Summary

Summary

- Computer Algebra: Sergeraert's Kenzo system.

Summary

- Computer Algebra: Sergeraert's Kenzo system.
- Before Computer Algebra: effective homology.

Summary

- Computer Algebra: Sergeraert's Kenzo system.
- Before Computer Algebra: effective homology.
- Beyond Computer Algebra: mechanized mathematics.

Summary

- Computer Algebra: Sergeraert's Kenzo system.
- Before Computer Algebra: effective homology.
- Beyond Computer Algebra: mechanized mathematics.
- Proving as humans: Isabelle/HOL.

Summary

- Computer Algebra: Sergeraert's Kenzo system.
- Before Computer Algebra: effective homology.
- Beyond Computer Algebra: mechanized mathematics.
- Proving as humans: Isabelle/HOL.
- Being constructive: Coq.

Summary

- Computer Algebra: Sergeraert's Kenzo system.
- Before Computer Algebra: effective homology.
- Beyond Computer Algebra: mechanized mathematics.
- Proving as humans: Isabelle/HOL.
- Being constructive: Coq.
- Keeping close to Kenzo: ACL2.

Summary

- Computer Algebra: Sergeraert's Kenzo system.
- Before Computer Algebra: effective homology.
- Beyond Computer Algebra: mechanized mathematics.
- Proving as humans: Isabelle/HOL.
- Being constructive: Coq.
- Keeping close to Kenzo: ACL2.
- Integrating all of that: fKenzo.

Summary

- Computer Algebra: Sergeraert's Kenzo system.
- Before Computer Algebra: effective homology.
- Beyond Computer Algebra: mechanized mathematics.
- Proving as humans: Isabelle/HOL.
- Being constructive: Coq.
- Keeping close to Kenzo: ACL2.
- Integrating all of that: fKenzo.
- Formalising mathematics: the European Project ForMath.

Summary

- Computer Algebra: Sergeraert's Kenzo system.
- Before Computer Algebra: effective homology.
- Beyond Computer Algebra: mechanized mathematics.
- Proving as humans: Isabelle/HOL.
- Being constructive: Coq.
- Keeping close to Kenzo: ACL2.
- Integrating all of that: fKenzo.
- Formalising mathematics: the European Project ForMath.
- Conclusions.

Summary

- Computer Algebra: Sergeraert's Kenzo system.
- Before Computer Algebra: effective homology.
- Beyond Computer Algebra: mechanized mathematics.
- Proving as humans: Isabelle/HOL.
- Being constructive: Coq.
- Keeping close to Kenzo: ACL2.
- Integrating all of that: fKenzo.
- Formalising mathematics: the European Project ForMath.
- Conclusions.
- Beyond conclusions.

Computer Algebra in Algebraic Topology

Computer Algebra in Algebraic Topology

- Kenzo

Computer Algebra in Algebraic Topology

- Kenzo
  1. Common Lisp program created by F. Sergeraert (since 1990)

Computer Algebra in Algebraic Topology

- Kenzo
  1. Common Lisp program created by F. Sergeraert (since 1990)
  2. Computes homology groups of simplicial sets...

Computer Algebra in Algebraic Topology

- Kenzo
  1. Common Lisp program created by F. Sergeraert (since 1990)
  2. Computes homology groups of simplicial sets...
  3. ...including infinite dimensional spaces
     (as iterated loop spaces)

Computer Algebra in Algebraic Topology

- Kenzo
  1. Common Lisp program created by F. Sergeraert (since 1990)
  2. Computes homology groups of simplicial sets. . .
  3. . . . including infinite dimensional spaces
     (as iterated loop spaces)
- Demo

Computer Algebra in Algebraic Topology

- Kenzo
  1. Common Lisp program created by F. Sergeraert (since 1990)
  2. Computes homology groups of simplicial sets...
  3. ...including infinite dimensional spaces
     (as iterated loop spaces)
- Demo
- Trusting Kenzo

Computer Algebra in Algebraic Topology

- Kenzo
  1. Common Lisp program created by F. Sergeraert (since 1990)
  2. Computes homology groups of simplicial sets...
  3. ...including infinite dimensional spaces
     (as iterated loop spaces)
- Demo
- Trusting Kenzo
  1. Example: $H_6(\Omega^3(P^\infty\mathbb{R}/P^3\mathbb{R});\mathbb{Z}) = (\mathbb{Z}/2\mathbb{Z})^{10}$

Computer Algebra in Algebraic Topology

- Kenzo
  1. Common Lisp program created by F. Sergeraert (since 1990)
  2. Computes homology groups of simplicial sets. . .
  3. . . . including infinite dimensional spaces
     (as iterated loop spaces)
- Demo
- Trusting Kenzo
  1. Example: $H_6(\Omega^3(P^\infty\mathbb{R}/P^3\mathbb{R});\mathbb{Z}) = (\mathbb{Z}/2\mathbb{Z})^{10}$
  2. Is it correct?

Sergeraert's effective homology (1/3)

Sergeraert's effective homology (1/3)

- Philosophy: don't pass too early to the quotient.

Sergeraert's effective homology (1/3)

- Philosophy: don't pass too early to the quotient.
- Technique: keep an explicit link (reduction) between the chain complex of an space and a chain complex of finite type.

Sergeraert's effective homology (1/3)

- Philosophy: don't pass too early to the quotient.
- Technique: keep an explicit link (reduction) between the chain complex of an space and a chain complex of finite type.
- A chain complex is $\{(C_n, d_n)\}_{n\in\mathbb{Z}}$, where each C_n is an abelian group, and each $d_n : C_n \rightarrow C_{n-1}$ is a homomorphism satisfying $d_{n+1} \circ d_n = 0, \forall n \in \mathbb{Z}$.

Sergeraert's effective homology (1/3)

- Philosophy: don't pass too early to the quotient.
- Technique: keep an explicit link (reduction) between the chain complex of an space and a chain complex of finite type.
- A chain complex is $\{(C_n, d_n)\}_{n \in \mathbb{Z}}$, where each C_n is an abelian group, and each $d_n : C_n \to C_{n-1}$ is a homomorphism satisfying $d_{n+1} \circ d_n = 0, \forall n \in \mathbb{Z}$.
- *Homology groups:* $H_n(C, d) := Ker(d_n)/Im(d_{n+1})$.

Sergeraert's effective homology (1/3)

- Philosophy: don't pass too early to the quotient.
- Technique: keep an explicit link (reduction) between the chain complex of an space and a chain complex of finite type.
- A chain complex is $\{(C_n, d_n)\}_{n\in\mathbb{Z}}$, where each C_n is an abelian group, and each $d_n : C_n \to C_{n-1}$ is a homomorphism satisfying $d_{n+1} \circ d_n = 0, \forall n \in \mathbb{Z}$.
- *Homology groups:* $H_n(C, d) := Ker(d_n)/Im(d_{n+1})$.
- Given two chain complexes $\{(C_n, d_n)\}_{n\in\mathbb{Z}}$ and $\{(C'_n, d'_n)\}_{n\in\mathbb{Z}}$, a *chain morphism* between them is a family f of group homomorphisms $f_n : C_n \to C'_n, \forall n \in \mathbb{Z}$ satisfying $d'_n \circ f_n = f_{n-1} \circ d_n, \forall n \in \mathbb{Z}$.

Sergeraert's effective homology (2/3)

Sergeraert's effective homology (2/3)

- Given two chain complexes $C := \{(C_n, d_n)\}_{n \in \mathbb{Z}}$ and $C' := \{(C'_n, d'_n)\}_{n \in \mathbb{Z}}$ a *reduction* between them is (f, g, h) where
  - $f : C \to C'$ and $g : C' \to C$ are chain morphisms
  - and h is a family of homomorphisms (called *homotopy operator*) $h_n : C_n \to C_{n+1}$.

  satisfying

Sergeraert's effective homology (2/3)

- Given two chain complexes $C := \{(C_n, d_n)\}_{n\in\mathbb{Z}}$ and $C' := \{(C'_n, d'_n)\}_{n\in\mathbb{Z}}$ a *reduction* between them is (f, g, h) where
  - $f : C \to C'$ and $g : C' \to C$ are chain morphisms
  - and h is a family of homomorphisms (called *homotopy operator*) $h_n : C_n \to C_{n+1}$.

  satisfying

  1. $f \circ g = 1$
  2. $d \circ h + h \circ d + g \circ f = 1$
  3. $f \circ h = 0$
  4. $h \circ g = 0$
  5. $h \circ h = 0$

Sergeraert's effective homology (2/3)

- Given two chain complexes $C := \{(C_n, d_n)\}_{n\in\mathbb{Z}}$ and $C' := \{(C'_n, d'_n)\}_{n\in\mathbb{Z}}$ a *reduction* between them is (f, g, h) where
  - $f : C \to C'$ and $g : C' \to C$ are chain morphisms
  - and h is a family of homomorphisms (called *homotopy operator*) $h_n : C_n \to C_{n+1}$.

  satisfying

  1. $f \circ g = 1$
  2. $d \circ h + h \circ d + g \circ f = 1$
  3. $f \circ h = 0$
  4. $h \circ g = 0$
  5. $h \circ h = 0$

- If $(f, g, h) : C \to C'$ is a reduction, then $H(C) \cong H(C')$.

Sergeraert's effective homology (3/3)

Sergeraert's effective homology (3/3)

- From now on, all the groups in chain complexes will be *free* abelian groups with an explicit basis.

Sergeraert's effective homology (3/3)

- From now on, all the groups in chain complexes will be *free* abelian groups with an explicit basis.
- A chain complex is *effective*, if $\forall n \in \mathbb{Z}, B_n$ is a finite set presented as a list of elements.

Sergeraert's effective homology (3/3)

- From now on, all the groups in chain complexes will be *free* abelian groups with an explicit basis.
- A chain complex is *effective*, if $\forall n \in \mathbb{Z}$, B_n is a finite set presented as a list of elements.
- On the contrary, a chain complex is called *locally effective* if the only known data on their bases are their characteristic functions and an equality test.

Sergeraert's effective homology (3/3)

- From now on, all the groups in chain complexes will be *free* abelian groups with an explicit basis.
- A chain complex is *effective*, if $\forall n \in \mathbb{Z}$, B_n is a finite set presented as a list of elements.
- On the contrary, a chain complex is called *locally effective* if the only known data on their bases are their characteristic functions and an equality test.
- A chain complex with *effective homology* is a data structure $[C, E, f, g, h]$ where C is a chain complex (possibly locally effective), E is an *effective* chain complex, and $(f, g, h) : C \to E$ is a reduction.

Basic Perturbation Lemma

Basic Perturbation Lemma

- Given a chain complex (C, d), a *perturbation* for it is a family ρ of group homomorphisms $\rho_n : C_n \rightarrow C_{n-1}$ such that $(C, d + \rho)$ is again a chain complex (that is to say: $(d + \rho) \circ (d + \rho) = 0$).

Basic Perturbation Lemma

- Given a chain complex (C, d), a *perturbation* for it is a family ρ of group homomorphisms $\rho_n : C_n \rightarrow C_{n-1}$ such that $(C, d + \rho)$ is again a chain complex (that is to say: $(d + \rho) \circ (d + \rho) = 0$).
- A reduction $(f, g, h) : (C, d) \rightarrow (C', d')$ and a perturbation ρ for (C, d) are *locally nilpotent* if
  $\forall x \in C_n, \exists m \in \mathbb{N}$ such that $(h \circ \rho)^m(x) = 0$.

Basic Perturbation Lemma

- Given a chain complex (C, d), a *perturbation* for it is a family ρ of group homomorphisms $\rho_n : C_n \longrightarrow C_{n-1}$ such that $(C, d + \rho)$ is again a chain complex (that is to say: $(d + \rho) \circ (d + \rho) = 0$).
- A reduction $(f, g, h) : (C, d) \rightarrow (C', d')$ and a perturbation ρ for (C, d) are *locally nilpotent* if
  $\forall x \in C_n, \exists m \in \mathbb{N}$ such that $(h \circ \rho)^m(x) = 0$.

Basic Perturbation Lemma

Let $(f, g, h) : (C, d) \rightarrow (C', d')$ be a reduction and be ρ a perturbation for (C, d) which are locally nilpotent. Then there exists a reduction $(f_\infty, g_\infty, h_\infty) : (C, d + \rho) \rightarrow (C', d'_\infty)$.

Basic Perturbation Lemma

- Given a chain complex (C, d), a *perturbation* for it is a family ρ of group homomorphisms $\rho_n : C_n \rightarrow C_{n-1}$ such that $(C, d + \rho)$ is again a chain complex (that is to say: $(d + \rho) \circ (d + \rho) = 0$).
- A reduction $(f, g, h) : (C, d) \rightarrow (C', d')$ and a perturbation ρ for (C, d) are *locally nilpotent* if $\forall x \in C_n, \exists m \in \mathbb{N}$ such that $(h \circ \rho)^m(x) = 0$.

Basic Perturbation Lemma

Let $(f, g, h) : (C, d) \rightarrow (C', d')$ be a reduction and be ρ a perturbation for (C, d) which are locally nilpotent. Then there exists a reduction $(f_\infty, g_\infty, h_\infty) : (C, d + \rho) \rightarrow (C', d'_\infty)$.

Basic Perturbation Lemma Algorithm

Given a chain complex (C, d) with effective homology and ρ a perturbation for it satisfying the local nilpotency condition, then $(C, d + \rho)$ is a chain complex with effective homology.

Mechanising Mathematics

Mechanising Mathematics

- Beyond Computer Algebra (beyond computing)

Mechanising Mathematics

- Beyond Computer Algebra (beyond computing)
- Three axes towards the mechanisation of mathematics:

Mechanising Mathematics

- Beyond Computer Algebra (beyond computing)
- Three axes towards the mechanisation of mathematics:
  1. Computing

Mechanising Mathematics

- Beyond Computer Algebra (beyond computing)
- Three axes towards the mechanisation of mathematics:
  1. Computing
  2. Proving (mechanized reasoning)

Mechanising Mathematics

- Beyond Computer Algebra (beyond computing)
- Three axes towards the mechanisation of mathematics:
  1. Computing
  2. Proving (mechanized reasoning)
  3. Communicating
     (remote access, user interfaces, interoperability...)

Mechanising Mathematics

- Beyond Computer Algebra (beyond computing)
- Three axes towards the mechanisation of mathematics:
  1. Computing
  2. Proving (mechanized reasoning)
  3. Communicating
     (remote access, user interfaces, interoperability...)
- Final aim: integrated computer aided mathematical tools

Mechanising Mathematics

- Beyond Computer Algebra (beyond computing)
- Three axes towards the mechanisation of mathematics:
  1. Computing
  2. Proving (mechanized reasoning)
  3. Communicating
     (remote access, user interfaces, interoperability...)
- Final aim: integrated computer aided mathematical tools
- In our concrete case: with an emphasis in Software Engineering (Program Verification)

Logic for the working mathematician: HOL

Logic for the working mathematician: HOL

- Isabelle/HOL is an interactive theorem proving environment.

Logic for the working mathematician: HOL

- Isabelle/HOL is an interactive theorem proving environment.
- Higher Order Logic (HOL) allows the modeller to translate the "by hand" proofs to the computer, in a "quite" direct way.

Logic for the working mathematician: HOL

- Isabelle/HOL is an interactive theorem proving environment.
- Higher Order Logic (HOL) allows the modeller to translate the "by hand" proofs to the computer, in a "quite" direct way.
- First milestone: Jesús Aransay's proof of the Basic Perturbation Lemma in Isabelle/HOL.

Logic for the working mathematician: HOL

- Isabelle/HOL is an interactive theorem proving environment.
- Higher Order Logic (HOL) allows the modeller to translate the "by hand" proofs to the computer, in a "quite" direct way.
- First milestone: Jesús Aransay's proof of the Basic Perturbation Lemma in Isabelle/HOL.
- Isabelle statement:

theorem *(**in** BPL) BPL:* **shows** *reduction D'*
$(\!|$ *carrier = carrier C, mult = mult C, one = one C, diff =* *(λx. if $x \in$ carrier C then (differ$_C$) $x \otimes_C (f \circ \delta \circ \Psi \circ g)\ x$ else $\mathbf{1}_C$)*$|\!)$ $(f \circ \Phi)$ $(\Psi \circ g)$ $(h \circ \Phi)$

Logic for the working mathematician: HOL

- Isabelle/HOL is an interactive theorem proving environment.
- Higher Order Logic (HOL) allows the modeller to translate the "by hand" proofs to the computer, in a "quite" direct way.
- First milestone: Jesús Aransay's proof of the Basic Perturbation Lemma in Isabelle/HOL.
- Isabelle statement:

  theorem *(**in** BPL) BPL:* **shows** *reduction D'*
  $(\!|$ *carrier = carrier C, mult = mult C, one = one C, diff =*
  $(\lambda x.$ *if* $x \in$ *carrier C then* $(differ_C)\ x \otimes_C (f \circ \delta \circ \Psi \circ g)\ x$
  else $\mathbf{1}_C)|\!)$ $\quad (f \circ \Phi)\ (\Psi \circ g)\ (h \circ \Phi)$

- Further challenge: program extraction.

The role of constructivism

The role of constructivism

- When working in constructive type theory: programming and proving become the same (Curry-Howard isomorphism)

The role of constructivism

- When working in constructive type theory:
  programming and proving become the same
  (Curry-Howard isomorphism)
- A proof assistant based on constructive type theory: Coq
  (Huet-Coquand Calculus of Constructions)

The role of constructivism

- When working in constructive type theory:
  programming and proving become the same
  (Curry-Howard isomorphism)
- A proof assistant based on constructive type theory: Coq
  (Huet-Coquand Calculus of Constructions)
- Benefits: programs for free.

The role of constructivism

- When working in constructive type theory:
  programming and proving become the same
  (Curry-Howard isomorphism)
- A proof assistant based on constructive type theory: Coq
  (Huet-Coquand Calculus of Constructions)
- Benefits: programs for free.
- Drawback: more demanding proofs.

The role of constructivism

- When working in constructive type theory:
  programming and proving become the same
  (Curry-Howard isomorphism)
- A proof assistant based on constructive type theory: Coq
  (Huet-Coquand Calculus of Constructions)
- Benefits: programs for free.
- Drawback: more demanding proofs.
- Only one example: the types C_i and C_{i+1-1} are different
  (with `i` integer).

The role of constructivism

- When working in constructive type theory: programming and proving become the same (Curry-Howard isomorphism)
- A proof assistant based on constructive type theory: Coq (Huet-Coquand Calculus of Constructions)
- Benefits: programs for free.
- Drawback: more demanding proofs.
- Only one example: the types C_i and C_{i+1-1} are different (with `i` integer).
- César Domínguez proved in Coq the effective homology of bicomplexes.

The role of constructivism

- When working in constructive type theory:
  programming and proving become the same
  (Curry-Howard isomorphism)
- A proof assistant based on constructive type theory: Coq
  (Huet-Coquand Calculus of Constructions)
- Benefits: programs for free.
- Drawback: more demanding proofs.
- Only one example: the types C_i and C_{i+1-1} are different
  (with `i` integer).
- César Domínguez proved in Coq the effective homology of
  bicomplexes.
- Distance from Kenzo?

Proving properties of Common Lisp programs: ACL2

Proving properties of Common Lisp programs: ACL2

- ACL2 = A Computational Logic for Applicative Common Lisp (ACL^2).

Proving properties of Common Lisp programs: ACL2

- ACL2 = A Computational Logic for Applicative Common Lisp (ACL^2).
- ACL2 is:

Proving properties of Common Lisp programs: ACL2

- ACL2 = A Computational Logic for Applicative Common Lisp (ACL^2).
- ACL2 is:
  - A programming language (an *applicative* subset of Common Lisp).

Proving properties of Common Lisp programs: ACL2

- ACL2 = A Computational Logic for Applicative Common Lisp (ACL^2).
- ACL2 is:
  - A programming language (an *applicative* subset of Common Lisp).
  - A logic (a restricted first-order one, with few quantifiers).

Proving properties of Common Lisp programs: ACL2

- ACL2 = A Computational Logic for Applicative Common Lisp (ACL^2).
- ACL2 is:
  - A programming language (an *applicative* subset of Common Lisp).
  - A logic (a restricted first-order one, with few quantifiers).
  - A theorem prover for that logic (on programs properties).

Proving properties of Common Lisp programs: ACL2

- ACL2 = A Computational Logic for Applicative Common Lisp (ACL^2).
- ACL2 is:
  - A programming language (an *applicative* subset of Common Lisp).
  - A logic (a restricted first-order one, with few quantifiers).
  - A theorem prover for that logic (on programs properties).
- Could Kenzo be verified in ACL2?

Proving properties of Common Lisp programs: ACL2

- ACL2 = A Computational Logic for Applicative Common Lisp (ACL^2).
- ACL2 is:
  - A programming language (an *applicative* subset of Common Lisp).
  - A logic (a restricted first-order one, with few quantifiers).
  - A theorem prover for that logic (on programs properties).
- Could Kenzo be verified in ACL2?
- ACL2 is first order...

Proving properties of Common Lisp programs: ACL2

- ACL2 = A Computational Logic for Applicative Common Lisp (ACL^2).
- ACL2 is:
  - A programming language (an *applicative* subset of Common Lisp).
  - A logic (a restricted first-order one, with few quantifiers).
  - A theorem prover for that logic (on programs properties).
- Could Kenzo be verified in ACL2?
- ACL2 is first order...
- ...but Kenzo intensively uses higher-order functional programming (functional coding of infinite sets).

Proving properties of Common Lisp programs: ACL2

- ACL2 = A Computational Logic for Applicative Common Lisp (ACL^2).
- ACL2 is:
  - A programming language (an *applicative* subset of Common Lisp).
  - A logic (a restricted first-order one, with few quantifiers).
  - A theorem prover for that logic (on programs properties).
- Could Kenzo be verified in ACL2?
- ACL2 is first order...
- ...but Kenzo intensively uses higher-order functional programming (functional coding of infinite sets).
- (Recall: Isabelle/HOL and Coq were higher order tools.)

Proving properties of Common Lisp programs: ACL2

- ACL2 = A Computational Logic for Applicative Common Lisp (ACL^2).
- ACL2 is:
  - A programming language (an *applicative* subset of Common Lisp).
  - A logic (a restricted first-order one, with few quantifiers).
  - A theorem prover for that logic (on programs properties).
- Could Kenzo be verified in ACL2?
- ACL2 is first order...
- ...but Kenzo intensively uses higher-order functional programming (functional coding of infinite sets).
- (Recall: Isabelle/HOL and Coq were higher order tools.)
- Pragmatic approach: ACL2 verification of *first order* fragments of Kenzo.

Proving properties of Kenzo in ACL2

Proving properties of Kenzo in ACL2

- Verifying that the implementation of Kenzo objects really corresponds with the actual mathematical objects.

Proving properties of Kenzo in ACL2

- Verifying that the implementation of Kenzo objects really corresponds with the actual mathematical objects.
- Example: build-in Kenzo simplicial sets are *really* simplicial sets.

Proving properties of Kenzo in ACL2

- Verifying that the implementation of Kenzo objects really corresponds with the actual mathematical objects.
- Example: build-in Kenzo simplicial sets are *really* simplicial sets.
- Verify that $\partial_i \partial_j = \partial_j \partial_{i+1}$ if $0 \leq j \leq i$
  Kenzo only provides a function computing ∂_i, but no way to *prove* that the function has the right properties.

Proving properties of Kenzo in ACL2

- Verifying that the implementation of Kenzo objects really corresponds with the actual mathematical objects.
- Example: build-in Kenzo simplicial sets are *really* simplicial sets.
- Verify that $\partial_i \partial_j = \partial_j \partial_{i+1}$ if $0 \leq j \leq i$
  Kenzo only provides a function computing ∂_i, but no way to *prove* that the function has the right properties.
- Developed in ACL2 by Jónathan Heras and Vico Pascual.

Proving properties of Kenzo in ACL2

- Verifying that the implementation of Kenzo objects really corresponds with the actual mathematical objects.
- Example: build-in Kenzo simplicial sets are *really* simplicial sets.
- Verify that $\partial_i \partial_j = \partial_j \partial_{i+1}$ if $0 \leq j \leq i$
  Kenzo only provides a function computing ∂_i, but no way to *prove* that the function has the right properties.
- Developed in ACL2 by Jónathan Heras and Vico Pascual.
- Difficulties to reach the *real* Kenzo code.

Proving properties of Kenzo in ACL2

- Verifying that the implementation of Kenzo objects really corresponds with the actual mathematical objects.
- Example: build-in Kenzo simplicial sets are *really* simplicial sets.
- Verify that $\partial_i \partial_j = \partial_j \partial_{i+1}$ if $0 \leq j \leq i$
  Kenzo only provides a function computing ∂_i, but no way to *prove* that the function has the right properties.
- Developed in ACL2 by Jónathan Heras and Vico Pascual.
- Difficulties to reach the *real* Kenzo code.
  - ACL2 is only a *subset* of Common Lisp
    (no loops, no destructive modifications).

Proving properties of Kenzo in ACL2

- Verifying that the implementation of Kenzo objects really corresponds with the actual mathematical objects.
- Example: build-in Kenzo simplicial sets are *really* simplicial sets.
- Verify that $\partial_i \partial_j = \partial_j \partial_{i+1}$ if $0 \leq j \leq i$
  Kenzo only provides a function computing ∂_i, but no way to *prove* that the function has the right properties.
- Developed in ACL2 by Jónathan Heras and Vico Pascual.
- Difficulties to reach the *real* Kenzo code.
  - ACL2 is only a *subset* of Common Lisp
    (no loops, no destructive modifications).
  - Poorer performance in the ACL2 version.

Proving properties of Kenzo in ACL2

- Verifying that the implementation of Kenzo objects really corresponds with the actual mathematical objects.
- Example: build-in Kenzo simplicial sets are *really* simplicial sets.
- Verify that $\partial_i \partial_j = \partial_j \partial_{i+1}$ if $0 \leq j \leq i$
  Kenzo only provides a function computing ∂_i, but no way to *prove* that the function has the right properties.
- Developed in ACL2 by Jónathan Heras and Vico Pascual.
- Difficulties to reach the *real* Kenzo code.
  - ACL2 is only a *subset* of Common Lisp
    (no loops, no destructive modifications).
  - Poorer performance in the ACL2 version.
- On-going research. . .

Proving simplicial theorems with ACL2

Proving simplicial theorems with ACL2

- ACL2 can be used to formalize mathematics, that, in principle, would need higher order logic.

Proving simplicial theorems with ACL2

- ACL2 can be used to formalize mathematics, that, in principle, would need higher order logic.
- Example: $\forall$ simplicial set K, there exists a *homotopy equivalence* between $C(K)$ the chain complex of K and $C^N(K)$, the *normalized* chain complex of K.

Proving simplicial theorems with ACL2

- ACL2 can be used to formalize mathematics, that, in principle, would need higher order logic.
- Example: $\forall$ simplicial set K, there exists a *homotopy equivalence* between $C(K)$ the chain complex of K and $C^N(K)$, the *normalized* chain complex of K.
  - Relation with Kenzo: this justifies the use in Kenzo of the smaller version $C^N(K)$.

Proving simplicial theorems with ACL2

- ACL2 can be used to formalize mathematics, that, in principle, would need higher order logic.
- Example: $\forall$ simplicial set K, there exists a *homotopy equivalence* between $C(K)$ the chain complex of K and $C^N(K)$, the *normalized* chain complex of K.
  - Relation with Kenzo: this justifies the use in Kenzo of the smaller version $C^N(K)$.
  - Developed in ACL2 by F. J. Martín-Mateos and J. L. Ruiz-Reina (Sevilla) and L. Lambán.

Proving simplicial theorems with ACL2

- ACL2 can be used to formalize mathematics, that, in principle, would need higher order logic.
- Example: $\forall$ simplicial set K, there exists a *homotopy equivalence* between $C(K)$ the chain complex of K and $C^N(K)$, the *normalized* chain complex of K.
  - Relation with Kenzo: this justifies the use in Kenzo of the smaller version $C^N(K)$.
  - Developed in ACL2 by F. J. Martín-Mateos and J. L. Ruiz-Reina (Sevilla) and L. Lambán.
- Going down to first order: more mathematics are needed.

Proving simplicial theorems with ACL2

- ACL2 can be used to formalize mathematics, that, in principle, would need higher order logic.
- Example: $\forall$ simplicial set K, there exists a *homotopy equivalence* between $C(K)$ the chain complex of K and $C^N(K)$, the *normalized* chain complex of K.
  - Relation with Kenzo: this justifies the use in Kenzo of the smaller version $C^N(K)$.
  - Developed in ACL2 by F. J. Martín-Mateos and J. L. Ruiz-Reina (Sevilla) and L. Lambán.
- Going down to first order: more mathematics are needed.
- In this concrete case: working with a category of pre-sheaves.

Integrating all of that: fKenzo.

Integrating all of that: fKenzo.

- Jónathan Heras's *fKenzo* = friendly Kenzo.

Integrating all of that: fKenzo.

- Jónathan Heras's *fKenzo* = friendly Kenzo.
- *fKenzo* is a Kenzo front-end and...

Integrating all of that: fKenzo.

- Jónathan Heras's *fKenzo* = friendly Kenzo.
- *fKenzo* is a Kenzo front-end and...
- ...an experimental tool for systems integration and interoperability.

Integrating all of that: fKenzo.

- Jónathan Heras's *fKenzo* = friendly Kenzo.
- *fKenzo* is a Kenzo front-end and...
- ...an experimental tool for systems integration and interoperability.
- OpenMath as interlingua.

Integrating all of that: fKenzo.

- Jónathan Heras's *fKenzo* = friendly Kenzo.
- *fKenzo* is a Kenzo front-end and...
- ...an experimental tool for systems integration and interoperability.
- OpenMath as interlingua.
- Example (Ana Romero): using GAP to compute group resolutions, and then Kenzo to deal with the corresponding aspherical simplicial space.

Integrating all of that: fKenzo.

- Jónathan Heras's *fKenzo* = friendly Kenzo.
- *fKenzo* is a Kenzo front-end and...
- ...an experimental tool for systems integration and interoperability.
- OpenMath as interlingua.
- Example (Ana Romero): using GAP to compute group resolutions, and then Kenzo to deal with the corresponding aspherical simplicial space.
- What about theorem proving?

Integrating all of that: fKenzo.

- Jónathan Heras's *fKenzo* = friendly Kenzo.
- *fKenzo* is a Kenzo front-end and...
- ...an experimental tool for systems integration and interoperability.
- OpenMath as interlingua.
- Example (Ana Romero): using GAP to compute group resolutions, and then Kenzo to deal with the corresponding aspherical simplicial space.
- What about theorem proving?
- Demo.

Formalising mathematics: the European Project ForMath

Formalising mathematics: the European Project ForMath

- European Commission FP7, STREP project ForMath: 2010-2013

Formalising mathematics: the European Project ForMath

- European Commission FP7, STREP project ForMath: 2010-2013
- Objective: formalized libraries for mathematical algorithms.

Formalising mathematics: the European Project ForMath

- European Commission FP7, STREP project ForMath: 2010-2013
- Objective: formalized libraries for mathematical algorithms.
- Four nodes:
  - Gothenburg University: Thierry Coquand, leader.
  - Radboud University.
  - INRIA.
  - Universidad de La Rioja.

Formalising mathematics: the European Project ForMath

- European Commission FP7, STREP project ForMath: 2010-2013
- Objective: formalized libraries for mathematical algorithms.
- Four nodes:
  - Gothenburg University: Thierry Coquand, leader.
  - Radboud University.
  - INRIA.
  - Universidad de La Rioja.
- Four Work Packages:

Formalising mathematics: the European Project ForMath

- European Commission FP7, STREP project ForMath: 2010-2013
- Objective: formalized libraries for mathematical algorithms.
- Four nodes:
  - Gothenburg University: Thierry Coquand, leader.
  - Radboud University.
  - INRIA.
  - Universidad de La Rioja.
- Four Work Packages:
  - Infrastructure to formalize mathematics in constructive type theory (`ssreflect`, Gonthier's mechanized proof of the Four Colour Theorem).

Formalising mathematics: the European Project ForMath

- European Commission FP7, STREP project ForMath: 2010-2013
- Objective: formalized libraries for mathematical algorithms.
- Four nodes:
  - Gothenburg University: Thierry Coquand, leader.
  - Radboud University.
  - INRIA.
  - Universidad de La Rioja.
- Four Work Packages:
  - Infrastructure to formalize mathematics in constructive type theory (`ssreflect`, Gonthier's mechanized proof of the Four Colour Theorem).
  - Linear Algebra library.

Formalising mathematics: the European Project ForMath

- European Commission FP7, STREP project ForMath: 2010-2013
- Objective: formalized libraries for mathematical algorithms.
- Four nodes:
  - Gothenburg University: Thierry Coquand, leader.
  - Radboud University.
  - INRIA.
  - Universidad de La Rioja.
- Four Work Packages:
  - Infrastructure to formalize mathematics in constructive type theory (`ssreflect`, Gonthier's mechanized proof of the Four Colour Theorem).
  - Linear Algebra library.
  - Real numbers and differential equations.

Formalising mathematics: the European Project ForMath

- European Commission FP7, STREP project ForMath: 2010-2013
- Objective: formalized libraries for mathematical algorithms.
- Four nodes:
  - Gothenburg University: Thierry Coquand, leader.
  - Radboud University.
  - INRIA.
  - Universidad de La Rioja.
- Four Work Packages:
  - Infrastructure to formalize mathematics in constructive type theory (`ssreflect`, Gonthier's mechanized proof of the Four Colour Theorem).
  - Linear Algebra library.
  - Real numbers and differential equations.
  - Algebraic topology and. . . (medical) image processing.

Conclusions

Conclusions

- Theorem Provers are mature enough to tackle *real* mathematical problems.

Conclusions

- Theorem Provers are mature enough to tackle *real* mathematical problems.
- Specially interesting in conjunction with Computer Algebra systems (increasing reliability).

Conclusions

- Theorem Provers are mature enough to tackle *real* mathematical problems.
- Specially interesting in conjunction with Computer Algebra systems (increasing reliability).
- Our subject: Algebraic Topology.

Conclusions

- Theorem Provers are mature enough to tackle *real* mathematical problems.
- Specially interesting in conjunction with Computer Algebra systems (increasing reliability).
- Our subject: Algebraic Topology.
  - Computing: Kenzo.

Conclusions

- Theorem Provers are mature enough to tackle *real* mathematical problems.
- Specially interesting in conjunction with Computer Algebra systems (increasing reliability).
- Our subject: Algebraic Topology.
  - Computing: Kenzo.
  - Proving: Isabelle, Coq, ACL2.

Conclusions

- Theorem Provers are mature enough to tackle *real* mathematical problems.
- Specially interesting in conjunction with Computer Algebra systems (increasing reliability).
- Our subject: Algebraic Topology.
  - Computing: Kenzo.
  - Proving: Isabelle, Coq, ACL2.
  - Integration: fKenzo.

Conclusions

- Theorem Provers are mature enough to tackle *real* mathematical problems.
- Specially interesting in conjunction with Computer Algebra systems (increasing reliability).
- Our subject: Algebraic Topology.
  - Computing: Kenzo.
  - Proving: Isabelle, Coq, ACL2.
  - Integration: fKenzo.
- Much more research effort is needed to devise a really usable and flexible tool.

Something that is not concluding...

Something that is not concluding...

- . . . my relationship with Tomás Recio.

Something that is not concluding...

- . . . my relationship with Tomás Recio.
- Geometry.
- Computer Algebra.
- Mathematical Education.

Something that is not concluding...

- ... my relationship with Tomás Recio.
- Geometry.
  - 1987, February, Preparata's talk, Zaragoza.
- Computer Algebra.
- Mathematical Education.

Something that is not concluding...

- . . . my relationship with Tomás Recio.
- Geometry.
  - 1987, February, Preparata's talk, Zaragoza.
  - 1987, September, conference at Sevilla.
- Computer Algebra.
- Mathematical Education.

Something that is not concluding...

- . . . my relationship with Tomás Recio.
- Geometry.
  - 1987, February, Preparata's talk, Zaragoza.
  - 1987, September, conference at Sevilla.
  - 1988, Institut Fourier, Grenoble.
- Computer Algebra.
- Mathematical Education.

Something that is not concluding...

- ...my relationship with Tomás Recio.
- Geometry.
  - 1987, February, Preparata's talk, Zaragoza.
  - 1987, September, conference at Sevilla.
  - 1988, Institut Fourier, Grenoble.
- Computer Algebra.
  - 1993, Pedro Real PhD thesis.
- Mathematical Education.

Something that is not concluding...

- . . . my relationship with Tomás Recio.
- Geometry.
  - 1987, February, Preparata's talk, Zaragoza.
  - 1987, September, conference at Sevilla.
  - 1988, Institut Fourier, Grenoble.
- Computer Algebra.
  - 1993, Pedro Real PhD thesis.
  - 1999, Spanish Conference on Computer Algebra (EACA)
- Mathematical Education.

Something that is not concluding...

- . . . my relationship with Tomás Recio.
- Geometry.
  - 1987, February, Preparata's talk, Zaragoza.
  - 1987, September, conference at Sevilla.
  - 1988, Institut Fourier, Grenoble.
- Computer Algebra.
  - 1993, Pedro Real PhD thesis.
  - 1999, Spanish Conference on Computer Algebra (EACA)
  - Since 2001, Scientific Committee EACA.
- Mathematical Education.

Something that is not concluding...

- . . . my relationship with Tomás Recio.
- Geometry.
  - 1987, February, Preparata's talk, Zaragoza.
  - 1987, September, conference at Sevilla.
  - 1988, Institut Fourier, Grenoble.
- Computer Algebra.
  - 1993, Pedro Real PhD thesis.
  - 1999, Spanish Conference on Computer Algebra (EACA)
  - Since 2001, Scientific Committee EACA.
  - . . .
- Mathematical Education.

Something that is not concluding...

- . . . my relationship with Tomás Recio.
- Geometry.
  - 1987, February, Preparata's talk, Zaragoza.
  - 1987, September, conference at Sevilla.
  - 1988, Institut Fourier, Grenoble.
- Computer Algebra.
  - 1993, Pedro Real PhD thesis.
  - 1999, Spanish Conference on Computer Algebra (EACA)
  - Since 2001, Scientific Committee EACA.
  - . . .
- Mathematical Education.
  - R&D project TutorMates (company Addlink).

Something that is not concluding...

- . . . my relationship with Tomás Recio.
- Geometry.
  - 1987, February, Preparata's talk, Zaragoza.
  - 1987, September, conference at Sevilla.
  - 1988, Institut Fourier, Grenoble.
- Computer Algebra.
  - 1993, Pedro Real PhD thesis.
  - 1999, Spanish Conference on Computer Algebra (EACA)
  - Since 2001, Scientific Committee EACA.
  - . . .
- Mathematical Education.
  - R&D project TutorMates (company Addlink).

Thanks, Tomás!