Proving with ACL2 the correctness of simplicial sets in the Kenzo system\textsuperscript{1}

Jónathan Heras \quad Vico Pascual \quad Julio Rubio

*Departamento de Matemáticas y Computación*
Universidad de La Rioja
Spain

20th International Symposium on Logic-Based Program Synthesis and Transformation
LOPSTR 2010, Hagenberg, Austria

\textsuperscript{1}Partially supported by Ministerio de Educación y Ciencia, project MTM2009-13842-C02-01, and by European Commission FP7, STREP project ForMath
Implementation of stacks
Introductory Example

- Implementation of stacks
- Prove the correctness of our implementation

Stack

push

pop
Introduction

Introductory Example

- Implementation of stacks
- Prove the correctness of our implementation
  - Model the problem

Stack

- push
- pop

(defun stack-p (stack)
  (consp stack))

(defun push (elem stack)
  (cons elem stack))

(defun pop (stack)
  (cdr stack))
Introduction

Introductory Example

- Implementation of stacks
- Prove the correctness of our implementation
  - Model the problem
  - Prove the properties about push and pop

\[ (\text{defthm push-pop}) \]

\[ (\text{implies (stack-p stack)}) \]

\[ (\text{equal (pop (push a stack)) stack)}) \]

\[ ... \]

J. Heras, V. Pascual, J. Rubio

Proving with ACL2 the correctness of simplicial sets in the Kenzo system
Introduction

Introductory Example

- Implementation of stacks
- Prove the correctness of our implementation
  - Model the problem
  - Prove the properties about push and pop

⇒ Our implementation of a stack is really a stack

(implies (stack-p stack)
  (equal (pop (push a stack))
    stack)))

...
The Kenzo system

- Kenzo:
The Kenzo system

Kenzo:
- Symbolic Computation System devoted to Algebraic Topology
The Kenzo system

- Kenzo:
  - Symbolic Computation System devoted to Algebraic Topology
  - Common Lisp package
The Kenzo system

- Kenzo:
  - Symbolic Computation System devoted to Algebraic Topology
  - Common Lisp package
  - Homology groups unreachable by any other means
The Kenzo system

- Kenzo:
  - Symbolic Computation System devoted to Algebraic Topology
  - Common Lisp package
  - Homology groups unreachable by any other means

General Goal

*Increase the reliability of the Kenzo system beyond testing*
The Kenzo system

- Kenzo:
  - Symbolic Computation System devoted to Algebraic Topology
  - Common Lisp package
  - Homology groups unreachable by any other means

General Goal

*Increase the reliability of the Kenzo system beyond testing*

- Isabelle/Hol and Coq:
The Kenzo system

- Kenzo:
  - Symbolic Computation System devoted to Algebraic Topology
  - Common Lisp package
  - Homology groups unreachable by any other means

General Goal

*Increase the reliability of the Kenzo system beyond testing*

- Isabelle/Hol and Coq:
  - Higher Order Logic
The Kenzo system

- Kenzo:
  - Symbolic Computation System devoted to Algebraic Topology
  - Common Lisp package
  - Homology groups unreachable by any other means

General Goal

*Increase the reliability of the Kenzo system beyond testing*

- Isabelle/Hol and Coq:
  - Higher Order Logic
  - Proofs related to algorithms
The Kenzo system

- Kenzo:
  - Symbolic Computation System devoted to Algebraic Topology
  - Common Lisp package
  - Homology groups unreachable by any other means

General Goal

*Increase the reliability of the Kenzo system beyond testing*

- Isabelle/Hol and Coq:
  - Higher Order Logic
  - Proofs related to algorithms
- ACL2:
The Kenzo system

- **Kenzo:**
  - Symbolic Computation System devoted to Algebraic Topology
  - Common Lisp package
  - Homology groups unreachable by any other means

**General Goal**

*Increase the reliability of the Kenzo system beyond testing*

- Isabelle/Hol and Coq:
  - Higher Order Logic
  - Proofs related to algorithms
- ACL2:
  - First Order Logic
The Kenzo system

- Kenzo:
  - Symbolic Computation System devoted to Algebraic Topology
  - Common Lisp package
  - Homology groups unreachable by any other means

General Goal

*Increase the reliability of the Kenzo system beyond testing*

- Isabelle/Hol and Coq:
  - Higher Order Logic
  - Proofs related to algorithms
- ACL2:
  - First Order Logic
  - Verification of real code
Current Work

- Kenzo way of working:

  1. Construction of constant spaces (spheres, Moore spaces, ...): \(\sim 20\%\)
  2. Construction of new spaces from other ones (cartesian products, loop spaces, ...): \(\sim 60\%\)
  3. Perform some computations (homology groups): \(\sim 10\%\)

Concrete Goal

Verify the correctness of Kenzo constructors of constant spaces

Kenzo first order logic fragments

Kenzo code

→ ACL2

Case Study

Each Kenzo Simplicial Set is really a simplicial set

J. Heras, V. Pascual, J. Rubio

Proving with ACL2 the correctness of simplicial sets in the Kenzo system
Current Work

- Kenzo way of working:
  - Construction of constant spaces (spheres, Moore spaces, ...): \( \sim 20\% \)
Current Work

- Kenzo way of working:
  1. Construction of constant spaces (spheres, Moore spaces, ...): $\sim 20\%$
  2. Construction of new spaces from other ones (cartesian products, loop spaces, ...): $\sim 60\%$

Concrete Goal

Verify the correctness of Kenzo constructors of constant spaces

Kenzo first order logic fragments
Kenzo code $\rightarrow$ ACL2

Case Study

Each Kenzo Simplicial Set is really a simplicial set

J. Heras, V. Pascual, J. Rubio
Current Work

- Kenzo way of working:
  1. Construction of constant spaces (spheres, Moore spaces, ...): \( \sim 20\% \)
  2. Construction of new spaces from other ones (cartesian products, loop spaces, ...): \( \sim 60\% \)
  3. Perform some computations (homology groups): \( \sim 10\% \)
Current Work

- Kenzo way of working:
  1. Construction of constant spaces (spheres, Moore spaces, ...): \(\sim 20\%\)
  2. Construction of new spaces from other ones (cartesian products, loop spaces, ...): \(\sim 60\%\)
  3. Perform some computations (homology groups): \(\sim 10\%\)

Concrete Goal

*Verify the correctness of Kenzo constructors of constant spaces*
Current Work

- Kenzo way of working:
  1. Construction of constant spaces (spheres, Moore spaces, ...): \( \sim 20\% \)
  2. Construction of new spaces from other ones (cartesian products, loop spaces, ...): \( \sim 60\% \)
  3. Perform some computations (homology groups): \( \sim 10\% \)

Concrete Goal

*Verify the correctness of Kenzo constructors of constant spaces*

- Kenzo first order logic fragments
Current Work

- Kenzo way of working:
  1. Construction of constant spaces (spheres, Moore spaces, ...): \( \sim 20\% \)
  2. Construction of new spaces from other ones (cartesian products, loop spaces, ...): \( \sim 60\% \)
  3. Perform some computations (homology groups): \( \sim 10\% \)

Concrete Goal

*Verify the correctness of Kenzo constructors of constant spaces*

- Kenzo first order logic fragments
- Kenzo code \( \rightarrow \) ACL2
Current Work

- Kenzo way of working:
  1. Construction of constant spaces (spheres, Moore spaces, ...): \( \sim 20\% \)
  2. Construction of new spaces from other ones (cartesian products, loop spaces, ...): \( \sim 60\% \)
  3. Perform some computations (homology groups): \( \sim 10\% \)

Concrete Goal

*Verify the correctness of Kenzo constructors of constant spaces*

- Kenzo first order logic fragments
- Kenzo code → ACL2

Case Study

*Each Kenzo Simplicial Set is really a simplicial set*
Table of Contents
Mathematical context: Simplicial Sets

Definition

A simplicial set $K$, is a union $K = \bigcup_{q \geq 0} K^q$, where the $K^q$ are disjoint sets, together with functions:

$$\partial^q_i : K^q \to K^{q-1}, \quad q > 0, \quad i = 0, \ldots, q,$$

$$\eta^q_i : K^q \to K^{q+1}, \quad q \geq 0, \quad i = 0, \ldots, q,$$

subject to the relations:

1. $\partial^{q-1}_i \partial^q_j = \partial^{q-1}_{j-1} \partial^q_i$ if $i < j$,
2. $\eta^{q+1}_i \eta^q_j = \eta^{q+1}_{j+1} \eta^q_i$ if $i \leq j$,
3. $\partial^{q+1}_i \eta^q_j = \eta^{q-1}_{j-1} \partial^q_i$ if $i < j$,
4. $\partial^{q+1}_i \eta^q_i = \text{identity} = \partial^{q+1}_{i+1} \eta^q_i$,
5. $\partial^{q+1}_i \eta^q_j = \eta^{q-1}_j \partial^q_{i-1}$ if $i > j + 1$,
Mathematical context: Simplicial Sets

Definition

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:

1. $\partial^{q-1}_i \partial^q_j = \partial^{q-1}_{j-1} \partial^q_i$ if $i < j$,
2. $\eta^{q+1}_i \eta^q_j = \eta^{q+1}_{i+1} \eta^q_i$ if $i \leq j$,
3. $\partial^{q+1}_i \eta^q_j = \eta^{q-1}_{j-1} \partial^q_i$ if $i < j$,
4. $\partial^{q+1}_i \eta^q_j = \text{identity} = \partial^{q+1}_{i+1} \eta^q_i$,
5. $\partial^{q+1}_i \eta^q_j = \eta^{q-1}_j \partial^q_{i-1}$ if $i > j + 1$,

The elements of $K^q$ are called $q$-simplexes.
A simplicial set \( K \), is a union \( K = \bigcup_{q \geq 0} K^q \), where the \( K^q \) are disjoints sets, together with functions:

\[
\partial_i^q : K^q \to K^{q-1}, \quad q > 0, \quad i = 0, \ldots, q,
\]

\[
\eta_i^q : K^q \to K^{q+1}, \quad q \geq 0, \quad i = 0, \ldots, q,
\]

subject to the relations:

1. \( \partial_i^{q-1} \partial_j^q = \partial_{j-1}^q \partial_i^{q-1} \) if \( i < j \),
2. \( \eta_i^{q+1} \eta_j^q = \eta_j^{q+1} \eta_i^q \) if \( i \leq j \),
3. \( \partial_i^{q+1} \eta_j^q = \eta_{j-1}^q \partial_i^{q+1} \) if \( i < j \),
4. \( \partial_i^{q+1} \eta_j^q = \text{identity} = \partial_{i+1}^{q+1} \eta_i^q \),
5. \( \partial_i^{q+1} \eta_j^q = \eta_{j+1}^q \partial_i^{q+1} \) if \( i > j + 1 \),

- The elements of \( K^q \) are called \( q \)-simplexes
- A \( q \)-simplex \( x \) is degenerate if \( x = \eta_i^{q-1} y \) for some simplex \( y \in K^{q-1} \)
Mathematical context: Simplicial Sets

Definition

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:

(1) $\partial^q_i \partial^q_j = \partial^q_j \partial^q_i$ if $i < j$,
(2) $\eta^q_i \eta^q_j = \eta^q_j \eta^q_i$ if $i \leq j$,
(3) $\partial^q_i \eta^q_j = \eta^q_{j-1} \partial^q_i$ if $i < j$,
(4) $\partial^q_i \eta^q_i = \text{identity} = \partial^q_{i+1} \eta^q_i$,
(5) $\partial^q_i \eta^q_j = \eta^q_{j-1} \partial^q_{i-1}$ if $i > j + 1$,

- The elements of $K^q$ are called $q$-simplexes
- A $q$-simplex $x$ is degenerate if $x = \eta^q_i y$ for some simplex $y \in K^{q-1}$
- Otherwise $x$ is called non-degenerate
Mathematical context: Example

- 0-simplexes: vertices: \((a), (b), (c), (d)\)
- non-degenerate 1-simplexes: edges:
  \((a b), (a c), (a d), (b c), (b d), (c d)\)
- non-degenerate 2-simplexes: (filled) triangles:
  \((a b c), (a b d), (a c d), (b c d)\)
- non-degenerate 3-simplexes: (filled) tetrahedra: \((a b c d)\)
Mathematical context: Example

- **0-simplexes**: vertices: 
  \( (a), (b), (c), (d) \)

- **non-degenerate 1-simplexes**: 
  edges: 
  \( (a\ b), (a\ c), (a\ d), (b\ c), (b\ d), (c\ d) \)

- **non-degenerate 2-simplexes**: 
  (filled) triangles: 
  \( (a\ b\ c), (a\ b\ d), (a\ c\ d), (b\ c\ d) \)

- **non-degenerate 3-simplexes**: 
  (filled) tetrahedra: \( (a\ b\ c\ d) \)

**face**: \( \partial_i(a\ b\ c) = \begin{cases} 
  (b\ c) & \text{if } i = 0 \\
  (a\ c) & \text{if } i = 1 \\
  (a\ b) & \text{if } i = 2 
\end{cases} \)

geometrical meaning
Mathematical context: Example

- **0-simplexes**: vertices: \((a), (b), (c), (d)\)
- **non-degenerate 1-simplexes**: edges: \((a\, b), (a\, c), (a\, d), (b\, c), (b\, d), (c\, d)\)
- **non-degenerate 2-simplexes**: (filled) triangles: \((a\, b\, c), (a\, b\, d), (a\, c\, d), (b\, c\, d)\)
- **non-degenerate 3-simplexes**: (filled) tetrahedra: \((a\, b\, c\, d)\)

**Face**:

\[
\partial_i(a\ b\ c) = \begin{cases} 
(b\ c) & \text{if } i = 0 \\
(a\ c) & \text{if } i = 1 \\
(a\ b) & \text{if } i = 2 
\end{cases}
\]

**Degeneracy**:

\[
\eta_i(a\ b\ c) = \begin{cases} 
(a\ a\ b\ c) & \text{if } i = 0 \\
(a\ b\ b\ c) & \text{if } i = 1 \\
(a\ b\ c\ c) & \text{if } i = 2 
\end{cases}
\]
Proposition

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:

$$x = \eta_{j_k} \cdots \eta_{j_1} y$$

with $y \in K^r$, $k = n - r \geq 0$, and $0 \leq j_1 < \cdots < j_k < n$. 
Mathematical context: abstract simplexes

Proposition

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:

$$x = \eta_{j_k} \cdots \eta_{j_1} y$$

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

- **abstract simplex:**
Mathematical context: abstract simplexes

Proposition

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:

$$x = \eta_{j_k} \cdots \eta_{j_1} y$$

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

- abstract simplex:
  - $(dgop \ gmsm) := \left\{ \begin{array}{l}
dgop \text{ is a strictly decreasing sequence of degeneracy maps} \\
gmsm \text{ is a geometric simplex} \\
\end{array} \right.$
Mathematical context: abstract simplexes

Proposition

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 $0 \leq j_1 < \cdots < j_k < n$.

- **abstract simplex:**
  - $(dgop \ gmsm) := \begin{cases} 
  \text{dgop is a strictly decreasing sequence of degeneracy maps} \\
  \text{gmsm is a geometric simplex}
  \end{cases}$

Examples:

- non-degenerate
  - $(a \ b)$
- abstract simplex
  - $(\emptyset \ (a \ b))$
**Mathematical context: abstract simplexes**

**Proposition**

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 $0 \leq j_1 < \cdots < j_k < n$.

- **abstract simplex:**
  - $(dgop, gmsm) := \{ dgop \text{ is a strictly decreasing sequence of degeneracy maps, } gmsm \text{ is a geometric simplex} \}$
  - **Examples:**
    - non-degenerate simplex $(a \ b)$
    - abstract simplex $(\emptyset \ (a \ b))$
    - degenerate simplex $(a \ a \ b \ c)$
    - abstract simplex $(\eta_0 \ (a \ b \ c))$
Mathematical context: face and degeneracy

- **degeneracy operator**: \( \eta^q_i(\text{gsm}) := (\eta^q_i \circ \text{gsm}) \)
Mathematical context: face and degeneracy

- degeneracy operator: \( \eta^q_i(dgop \ gmsm) := (\eta^q_i \circ dgop \ gmsm) \)
- Independent from the simplicial set
Mathematical context: face and degeneracy

- **degeneracy operator:** \( \eta^q_i(dgop \ gmsm) := (\eta^q_i \circ dgop \ gmsm) \)
  
- Independent from the simplicial set
  
- \( \eta_2(\eta_3\eta_1 (a \ b \ c)) = (\eta_2\eta_3\eta_1 (a \ b \ c)) \) if \( i \leq j \)
  
- \( \eta_4\eta_2\eta_1 (a \ b \ c) \)
Mathematical context: face and degeneracy

- **degneracy operator**: $\eta_i^q(dgop \ gmsm) := (\eta_i^q \circ dgop \ gmsm)$
  - Independent from the simplicial set
  - $\eta_2(\eta_3\eta_1(a b c)) = (\eta_2\eta_3\eta_1(a b c))$ if $i \leq j$

- **face operator**:

$$\partial_i^q(dgop \ gmsm) := \begin{cases} (\partial_i^q \circ dgop \ gmsm) & \text{if } \eta_i \in dgop \lor \eta_{i-1} \in dgop \\ (\partial_i^q \circ dgop \ \partial_k^r gmsm) & \text{otherwise} \\ \end{cases}$$

where

- $r = q - \{\text{number of degeneracies in } dgop\}$ and
- $k = i - \{\text{number of degeneracies in } dgop \text{ with index lower than } i\}$
Mathematical context: face and degeneracy

- **degeneracy operator:** \( \eta_i^q(dgop \ gmsm) := (\eta_i^q \circ dgop \ gmsm) \)
  - Independent from the simplicial set
  - \( \eta_2(\eta_3\eta_1 (a \ b \ c)) = (\eta_2\eta_3\eta_1 (a \ b \ c)) \) if \( i \leq j \)

- **face operator:**
  \[
  \partial_i^q(dgop \ gmsm) := \begin{cases} 
  (\partial_i^q \circ dgop \ gmsm) & \text{if } \eta_i \in dgop \lor \eta_{i-1} \in dgop \\
  (\partial_i^q \circ dgop \ \partial_r^r gmsm) & \text{otherwise;}
  \end{cases}
  \]

where

\[
\begin{align*}
  r &= q - \{\text{number of degeneracies in } dgop\} \text{ and} \\
  k &= i - \{\text{number of degeneracies in } dgop \text{ with index lower than } i\}
\end{align*}
\]

- Dependent from the simplicial set . . .
Mathematical context: face and degeneracy

- **degeneracy operator:** \( \eta_i^q(\text{dgop } \text{gmsm}) := (\eta^q_i \circ \text{dgop } \text{gmsm}) \)
  - Independent from the simplicial set
  - \( \eta_2(\eta_3\eta_1 (a \ b \ c)) = (\eta_2\eta_3\eta_1 (a \ b \ c)) \)
  - \( \eta_i \eta_j = \eta_{j+1} \eta_i \) if \( i \leq j \)
  - \( \eta_4 \eta_2 \eta_1 (a \ b \ c) \)

- **face operator:**
  \[
  \partial_i^q(\text{dgop } \text{gmsm}) := \begin{cases} 
  (\partial^q_i \circ \text{dgop } \text{gmsm}) & \text{if } \eta_i \in \text{dgop} \lor \eta_{i-1} \in \text{dgop} \\
  (\partial^q_i \circ \text{dgop } \partial^r_k \text{gmsm}) & \text{otherwise};
  \end{cases}
  \]

where

- \( r = q - \{\text{number of degeneracies in } \text{dgop}\} \) and
- \( k = i - \{\text{number of degeneracies in } \text{dgop} \text{ with index lower than } i\} \)

- Dependent from the simplicial set . . .

- but some parts are independent
Mathematical context: face and degeneracy

**degeneracy operator:** $\eta^q_i (dgop \ gmsm) := (\eta^q_i \circ dgop \ gmsm)$

- Independent from the simplicial set
- $\eta_2 (\eta_3 \eta_1 (a \ b \ c)) = (\eta_2 \eta_3 \eta_1 (a \ b \ c)) \eta_i \eta_j = \eta_{j+1} \eta_i \quad \text{if } i \leq j$\quad (\eta_4 \eta_2 \eta_1 (a \ b \ c))$

**face operator:**

$d\eta_i^q (dgop \ gmsm) := \begin{cases}
(\partial^q_i \circ dgop \ gmsm) & \text{if } \eta_i \in dgop \lor \eta_{i-1} \in dgop \\
(\partial^q_i \circ dgop \partial^r_k gmsm) & \text{otherwise;}
\end{cases}$

where

$r = q - \{\text{number of degeneracies in } dgop\}$ and

$k = i - \{\text{number of degeneracies in } dgop \text{ with index lower than } i\}$

- Dependent from the simplicial set . . .
- but some parts are independent

$\partial_2 (\eta_3 \eta_1 (a \ b \ c)) = (\partial_2 \eta_3 \eta_1 (a \ b \ c)) \partial_i \eta_j = \eta_{j-1} \partial_i \quad \text{if } i < j$

$\partial_{i+1} \eta_i = \text{identity}$\quad (\eta_2 (a \ b \ c))$
Mathematical context: face and degeneracy

- **degeneracy operator**: $\eta^q_i (dgop \ gmsm) := (\eta^q_i \circ dgop \ gmsm)$
  - Independent from the simplicial set
  - $\eta_2 (\eta_3 \eta_1 (a b c)) = (\eta_2 \eta_3 \eta_1 (a b c))$ if $i \leq j$

- **face operator**:

  $\partial^q_i (dgop \ gmsm) := \begin{cases} 
  (\partial^q_i \circ dgop \ gmsm) & \text{if } \eta_i \in dgop \lor \eta_{i-1} \in dgop \\
  (\partial^q_i \circ dgop \ \partial^r_k gmsm) & \text{otherwise};
  \end{cases}$

where

$r = q - \{\text{number of degeneracies in } dgop\}$ and
$k = i - \{\text{number of degeneracies in } dgop \text{ with index lower than } i\}$

- Dependent from the simplicial set . . .
- but some parts are independent

  - $\partial_2 (\eta_3 \eta_1 (a b c)) = (\partial_2 \eta_3 \eta_1 (a b c))$ if $i < j$
  - $\partial_i \eta_j = \eta_{j-1} \partial_i$ if $i < j$
  - $i+1 \eta_i = \text{identity}$

  - $\partial_2 (\eta_3 \eta_0 (a b c)) = (\partial_2 \eta_3 \eta_0 \partial_1 (a b c))$ if $i < j$
  - $\partial_i \eta_j = \eta_{j-1} \partial_i$ if $i < j$
  - $i+1 \eta_i = \text{identity}$
Mathematical context: minimal conditions

Theorem

Let the object \( \{K^q, \partial^q\}_{q \geq 0} \) such that for all element \( gmsm \in K^q \) the following properties hold:

1. \( \forall i, j \in \mathbb{N} : i < j \leq q \implies \partial_i^{q-1}(\partial_j^q gmsm) = \partial_j^{q-1}(\partial_i^q gmsm) \),

2. \( \forall i \in \mathbb{N}, i \leq q : \partial_i^q gmsm \in K^{q-1} \),

then \( \{K^q, \partial^q, \eta^q\}_{q \geq 0} \) is a simplicial set.
ACL2 framework: minimal conditions

**Theorem**

Let the object \( \{K^q, \partial^q\}_{q \geq 0} \) such that for all element \( gmsm \in K^q \) the following properties hold:

1. \( \forall i, j \in \mathbb{N} : i < j \leq q \implies \partial_i^q^{-1}(\partial_j^q gmsm) = \partial_j^q gmsm \)
2. \( \forall i \in \mathbb{N}, i \leq q : \partial_i^q gmsm \in K^{q-1} \)

then \( \{K^q, \partial^q, \eta^q\}_{q \geq 0} \) is a simplicial set
ACL2 framework: minimal conditions

**Theorem**

Let the object \( \{ K^q, \partial^q \} \geq 0 \) such that for all element \( gmsm \in K^q \) the following properties hold:

1. \( \forall i, j \in \mathbb{N}: i < j \leq q \implies \partial^q_{i-1}(\partial^q_j gmsm) = \partial^q_{j-1}(\partial^q_i gmsm) \),

2. \( \forall i \in \mathbb{N}, i \leq q: \partial^q_i gmsm \in K^{q-1} \),

then \( \{ K^q, \partial^q, \eta^q \} \geq 0 \) is a simplicial set.

(encapsulate
  ; Signatures
  (((face * * *) => *)
   ((dimension *) => *)
   ((canonical *) => *)
   ((inv-ss * *) => *))
  ...)
)

---

J. Heras, V. Pascual, J. Rubio
Proving with ACL2 the correctness of simplicial sets in the Kenzo system

12/1
ACL2 framework: minimal conditions

Theorem

Let the object \( \{K^q, \partial^q\}_{q \geq 0} \) such that for all element \( g_{msm} \in K^q \) the following properties hold:

1. \( \forall i, j \in \mathbb{N} : i < j \leq q \implies \partial_i^{q-1}(\partial^q_j \ g_{msm}) = \partial_{j-1}^{q-1}(\partial^q_i \ g_{msm}) \),

2. \( \forall i \in \mathbb{N}, i \leq q : \partial^q_i \ g_{msm} \in K^q-1 \),

then \( \{K^q, \partial^q, \eta^q\}_{q \geq 0} \) is a simplicial set.

(encapsulate
  ; Signatures
  ((face * * *) => *)
  ((dimension *) => *)
  ((canonical *) => *)
  ((inv-ss * *) => *)
  ; Theorems
  (defthm faceoface
    (implies (and (natp i) (natp j) (< i j) (inv-ss ss ls))
      (equal (face ss i (face ss j ls)) (face ss (- j 1) (face ss i ls))))))
ACL2 framework: minimal conditions

Theorem

Let the object \( \{K^q, \partial^q\}_{q \geq 0} \) such that for all element \( gmsm \in K^q \) the following properties hold:

1. \( \forall i, j \in \mathbb{N} : i < j \leq q \implies \partial^q_{i-1}(\partial^q_j gmsm) = \partial^q_{j-1}(\partial^q_i gmsm) \),

2. \( \forall i \in \mathbb{N}, i \leq q: \partial^q_i gmsm \in K^{q-1} \),

then \( \{K^q, \partial^q, \eta^q\}_{q \geq 0} \) is a simplicial set.

(jencapsulate
; Signatures
((face * * *) => *)
((dimension *) => *)
((canonical *) => *)
((inv-ss * *) => *)
; Theorems
(defthm faceoface
  (implies (and (natp i) (natp j) (< i j) (inv-ss ss ls))
    (equal (face ss i (face ss j ls)) (face ss (- j 1) (face ss i ls)))))
(defthm inv-ss-prop
  (implies (and (canonical absm) (natp i) (< i (dimension absm)))
    (equal (dimension (face ss i absm)) (1- (dimension absm)))
  )
; Witness ... )

J. Heras, V. Pascual, J. Rubio
Proving with ACL2 the correctness of simplicial sets in the Kenzo system
ACL2 framework: face and degeneracy

**Theorem**

Let the object \( \{ K^q, \partial^q \} _{q \geq 0} \) such that for all element \( gmsm \in K^q \) the following properties hold:

1. \( \forall i, j \in \mathbb{N} : i < j \leq q \implies \partial^q_{i-1}(\partial^q_j gmsm) = \partial^q_{j-1}(\partial^q_i gmsm) \),

2. \( \forall i \in \mathbb{N}, i \leq q : \partial^q_i gmsm \in K^{q-1} \),

then \( \{ K^q, \partial^q, \eta^q \} _{q \geq 0} \) is a simplicial set.

(defun imp-face-Kenzo (ss i q (dgop gmsm))
  (if (face-absm-dgop i dgop)
      (list (face-absm-dgop i dgop) gmsm)
      (list (face-absm-dgop i dgop) (face ss (face-absm-indx i dgop) gmsm))))

(defun imp-degeneracy-Kenzo (ss i q (dgop gmsm))
  (list (degeneracy-absm-dgop-dgop i dgop) gmsm))

(defun imp-inv-Kenzo (ss q (dgop gmsm))
  ...)

imp-inv-Kenzo is the characteristic function
ACL2 framework: Proof of Theorem

Let the object \( \{K^q, \partial^q\}_{q \geq 0} \) such that for all element \( gmsm \in K^q \) the following properties hold:

1. \( \forall i, j \in \mathbb{N} : i < j \leq q \implies \partial_{i}^{q-1}(\partial_{j}^{q}gmsm) = \partial_{j-1}^{q-1}(\partial_{i}^{q}gmsm), \)

2. \( \forall i \in \mathbb{N}, i \leq q: \partial_{i}^{q}gmsm \in K^{q-1} \),

then \( \{K^q, \partial^q, \eta^q\}_{q \geq 0} \) is a simplicial set

- imp-face-Kenzo and imp-degeneracy-Kenzo are well-defined

(defthm theorem-1
  (implies (imp-inv-Kenzo ss q (dgop gmsm))
    (imp-inv-Kenzo ss (1- q) (imp-face-Kenzo ss i q (dgop gmsm))))))
ACL2 framework: Proof of Theorem

Theorem

Let the object \( \{K^q, \partial^q\}_{q \geq 0} \) such that for all element \( gmsm \in K^q \) the following properties hold:

1. \( \forall i, j \in \mathbb{N} : i < j \leq q \implies \partial^q_{i-1}(\partial^q_j gmsm) = \partial^q_{j-1}(\partial^q_i gmsm) \),
2. \( \forall i \in \mathbb{N}, i \leq q : \partial^q_i gmsm \in K^{q-1} \),

then \( \{K^q, \partial^q, \eta^q\}_{q \geq 0} \) is a simplicial set

- \textbf{imp-face-Kenzo} and \textbf{imp-degeneracy-Kenzo} are well-defined

\[
\text{(defthm theorem-1)}
\]

\[
\text{(implies (imp-inv-Kenzo ss q (dgop gmsm))}
\]

\[
\text{(imp-inv-Kenzo ss (1- q) (imp-face-Kenzo ss i q (dgop gmsm)))})
\]

- \textbf{imp-face-Kenzo} and \textbf{imp-degeneracy-Kenzo} satisfy the 5 properties of simplicial sets

\[
\text{(defthm theorem-3)}
\]

\[
\text{(implies (and (imp-inv-Kenzo ss q (dgop gmsm)) (natp i) (natp j) (< i j))}
\]

\[
\text{(equal (imp-face-Kenzo ss i (1- q) (imp-face-Kenzo ss j q (dgop gmsm)))}
\]

\[
\text{(imp-face-Kenzo ss (1- j) (1- q) (imp-face-Kenzo ss i q (dgop gmsm)))})
\]
Methodological approach imported from:

Sketch of the proofs

Methodological approach imported from:


1. Prove each theorem with EAT representation
Sketch of the proofs

Methodological approach imported from:


1. Prove each theorem with EAT representation
   - EAT is the predecessor of Kenzo
Methodological approach imported from:


1. Prove each theorem with EAT representation
   - EAT is the predecessor of Kenzo
   - Implements the same ideas
Sketch of the proofs

Methodological approach imported from:


1. Prove each theorem with EAT representation
   - EAT is the predecessor of Kenzo
   - Implements the same ideas
   - Closer to mathematical representation
Sketch of the proofs

Methodological approach imported from:


1. Prove each theorem with EAT representation
   - EAT is the predecessor of Kenzo
   - Implements the same ideas
   - Closer to mathematical representation

2. Prove the equivalence between Kenzo and EAT functions module a domain transformation

\[
\begin{align*}
\text{imp-face-eat} & \iff \text{imp-face-Kenzo} \\
\text{imp-degeneracy-eat} & \iff \text{imp-degeneracy-Kenzo} \\
\text{imp-inv-eat} & \iff \text{imp-inv-Kenzo}
\end{align*}
\]
Methodological approach imported from:


1 Prove each theorem with EAT representation
   - EAT is the predecessor of Kenzo
   - Implements the same ideas
   - Closer to mathematical representation

2 Prove the equivalence between Kenzo and EAT functions
   module a domain transformation

\[
\begin{align*}
\text{imp-face-eat} & \iff \text{imp-face-Kenzo} \\
\text{imp-degeneracy-eat} & \iff \text{imp-degeneracy-Kenzo} \\
\text{imp-inv-eat} & \iff \text{imp-inv-Kenzo}
\end{align*}
\]

⇒ All the theorems are proved with Kenzo representation
Schema of the proof

Sketch of the proof

EAT/Kenzo representation

Example:

\[(\eta_3 \eta_1 (a \ b \ c)) \Rightarrow ((3 \ 1) \ (a \ b \ c))\]

EAT

Kenzo

J. Heras, V. Pascual, J. Rubio

Proving with ACL2 the correctness of simplicial sets in the Kenzo system
EAT/Kenzo representation

**EAT**
- abstract simplexes:

\[
(dgop \ gmsm) := \begin{cases} 
  dgop \text{ is a strictly decreasing list} \\
  gmsm \text{ is an object}
\end{cases}
\]

Example:

\[
(\eta_3 \eta_1 (a \ b \ c)) \rightsquigarrow ((3 \ 1) (a \ b \ c))
\]

**Kenzo**
- abstract simplexes:

\[
(dgop \ gmsm) := \begin{cases} 
  dgop \text{ is a natural number} \\
  gmsm \text{ is an object}
\end{cases}
\]

Example:

\[
(\eta_3 \eta_1 (a \ b \ c)) \rightsquigarrow (10 (a \ b \ c))
\]

\[
\eta_3 \eta_1 \rightsquigarrow (0 \ 1 \ 0 \ 1) \rightsquigarrow \\
0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 = 10
\]
EAT/Kenzo representation

**EAT**

- **abstract simplexes:**
  
  \[(dgop \ gmsm) := \begin{cases} 
  dgop & \text{is a strictly decreasing list} \\
  gmsm & \text{is an object} \end{cases} \]

  Example:
  
  \[(\eta_3 \eta_1 \ (a \ b \ c)) \leadsto ((3 \ 1) \ (a \ b \ c))\]

- **face, degeneracy:**
  implemented with recursive functions

**Kenzo**

- **abstract simplexes:**
  
  \[(dgop \ gmsm) := \begin{cases} 
  dgop & \text{is a natural number} \\
  gmsm & \text{is an object} \end{cases} \]

  Example:
  
  \[(\eta_3 \eta_1 \ (a \ b \ c)) \leadsto (10 \ (a \ b \ c))\]

  \[\eta_3 \eta_1 \leadsto (0 \ 1 \ 0 \ 1) \leadsto 0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 = 10\]

- **face, degeneracy:**
  implemented using efficient primitives dealing with binary numbers
**EAT/Kenzo representation**

**EAT**
- abstract simplexes:
  \[(dgop \ gmsm) := \begin{cases} \text{dgop is a strictly decreasing list} \\ \text{gmsm is an object} \end{cases}\]

Example:

\[(\eta_3 \eta_1 (a \ b \ c)) \leadsto ((3 \ 1) \ (a \ b \ c))\]

- face, degeneracy:
  implemented with recursive functions

- inefficient

- easy to prove

**Kenzo**
- abstract simplexes:
  \[(dgop \ gmsm) := \begin{cases} \text{dgop is a natural number} \\ \text{gmsm is an object} \end{cases}\]

Example:

\[(\eta_3 \eta_1 (a \ b \ c)) \leadsto (10 \ (a \ b \ c))\]

\[\eta_3 \eta_1 \leadsto (0 \ 1 \ 0 \ 1) \leadsto 0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 = 10\]

- face, degeneracy:
  implemented using efficient primitives dealing with binary numbers

- efficient

- difficult to prove
Proof of a theorem

We want to prove

(defthm theorem-3-Kenzo
  (implies (and (imp-inv-Kenzo ss q (dgop gmsm)) (natp i) (natp j) (< i j))
    (equal (imp-face-Kenzo ss i (1- q) (imp-face-Kenzo ss j q (dgop gmsm)))
      (imp-face-Kenzo ss (1- j) (1- q) (imp-face-Kenzo ss i q (dgop gmsm))))))
Proof of a theorem

We want to prove

\[
\text{(defthm theorem-3-Kenzo} \\
\text{(implies (and (imp-inv-Kenzo ss q (dgop gmsm)) (natp i) (natp j) (< i j))} \\
\text{\quad (equal (imp-face-Kenzo ss i (1- q) (imp-face-Kenzo ss j q (dgop gmsm)))} \\
\text{\quad \quad (imp-face-Kenzo ss (1- j) (1- q) (imp-face-Kenzo ss i q (dgop gmsm)))))}
\]

First we prove

\[
\text{(defthm theorem-3-eat} \\
\text{(implies (and (imp-inv-eat ss q (dgop gmsm)) (natp i) (natp j) (< i j))} \\
\text{\quad (equal (imp-face-eat ss i (1- q) (imp-face-eat ss j q (dgop gmsm)))} \\
\text{\quad \quad (imp-face-eat ss (1- j) (1- q) (imp-face-eat ss i q (dgop gmsm))))})
\]
Proof of a theorem

We want to prove

\[(\text{defthm theorem-3-Kenzo})\]

\[(\text{implies} (\text{and} (\text{imp-inv-Kenzo} \ ss \ q \ (\text{dgop gmsm})) (\text{natp} \ i) (\text{natp} \ j) (< i j))
  
  (\text{equal} (\text{imp-face-Kenzo} \ ss \ i \ (1- q) (\text{imp-face-Kenzo} \ ss \ j \ q \ (\text{dgop gmsm})))
  
  (\text{imp-face-Kenzo} \ ss \ (1- j) \ (1- q) (\text{imp-face-Kenzo} \ ss \ i \ q \ (\text{dgop gmsm}))))\)

First we prove

\[(\text{defthm theorem-3-eat})\]

\[(\text{implies} (\text{and} (\text{imp-inv-eat} \ ss \ q \ (\text{dgop gmsm})) (\text{natp} \ i) (\text{natp} \ j) (< i j))
  
  (\text{equal} (\text{imp-face-eat} \ ss \ i \ (1- q) (\text{imp-face-eat} \ ss \ j \ q \ (\text{dgop gmsm})))
  
  (\text{imp-face-eat} \ ss \ (1- j) \ (1- q) (\text{imp-face-eat} \ ss \ i \ q \ (\text{dgop gmsm}))))\)

- induction
- simplification
- study of cases
then we prove
\[ \text{imp-face-eat} \iff \text{imp-face-Kenzo} \]
then we prove
\[ \text{imp-face-eat} \iff \text{imp-face-Kenzo} \]

Difficult to prove
- Kenzo and EAT deal with different representations
- Kenzo implementation is not intuitive
Proof of a theorem continued

2 then we prove

\[ \text{imp-face-eat} \iff \text{imp-face-Kenzo} \]

- Difficult to prove
  - Kenzo and EAT deal with different representations
  - Kenzo implementation is not intuitive
- Definition of an intermediary representation
then we prove

\[ \text{imp-face-eat} \iff \text{imp-face-Kenzo} \]

- **Difficult to prove**
  - Kenzo and EAT deal with different representations
  - Kenzo implementation is not intuitive

- **Definition of an intermediary representation**
  - based on binary lists

<table>
<thead>
<tr>
<th>mathematical</th>
<th>EAT</th>
<th>Binary</th>
<th>Kenzo</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \eta_3 \eta_1 )</td>
<td>(3 1)</td>
<td>(0 1 0 1)</td>
<td>10</td>
</tr>
</tbody>
</table>

J. Heras, V. Pascual, J. Rubio

Proving with ACL2 the correctness of simplicial sets in the Kenzo system
then we prove

\[ \text{imp-face-eat} \iff \text{imp-face-Kenzo} \]

- Difficult to prove
  - Kenzo and EAT deal with different representations
  - Kenzo implementation is not intuitive

- Definition of an intermediary representation
  - based on binary lists

<table>
<thead>
<tr>
<th></th>
<th>EAT</th>
<th>Binary</th>
<th>Kenzo</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \eta_3 \eta_1 )</td>
<td>(3 1)</td>
<td>(0 1 0 1)</td>
<td>10</td>
</tr>
</tbody>
</table>

- Definition of imp-face-binary
  - Works with binary lists
  - Inspired from Kenzo functions
Proof of a theorem continued

2. Then we prove

\[ \text{imp-face-eat} \iff \text{imp-face-Kenzo} \]

- Difficult to prove
  - Kenzo and EAT deal with different representations
  - Kenzo implementation is not intuitive

- Definition of an intermediary representation
  - Based on binary lists

<table>
<thead>
<tr>
<th>mathematical</th>
<th>EAT</th>
<th>Binary</th>
<th>Kenzo</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \eta_3 \eta_1 )</td>
<td>(3, 1)</td>
<td>(0 1 0 1)</td>
<td>10</td>
</tr>
</tbody>
</table>

- Definition of \text{imp-face-binary}
  - Works with binary lists
  - Inspired from Kenzo functions

\[ \text{imp-face-eat} \iff \text{imp-face-binary} \iff \text{imp-face-Kenzo} \]
Distance from ACL2 code to actual Kenzo code: values

Kenzo

(defun 1dlop-dgop (1dlop dgop)
 (progn
   (when (logbitp 1dlop dgop)
     (let ((share (ash -1 1dlop)))
       (values
         (logxor
           (logand share (ash dgop -1))
           (logandc1 share dgop))
         nil)))
   (when (and (plusp 1dlop)
               (logbitp (1- 1dlop) dgop))
     (let ((share (ash -1 1dlop)))
       (setf share (ash share -1))
       (return-from 1dlop-dgop
         (values
           (logxor
             (logand share (ash dgop -1))
             (logandc1 share dgop))
           nil)))))
   (let ((share (ash -1 1dlop)))
     (let ((right (logandc1 share dgop)))
       (values
         (logxor
           right
           (logand share (ash dgop -1)))
         (- 1dlop (logcount right)))))))

ACL2

(defun 1dlop-dgop-dgop (1dlop dgop)
 (if (and (natp 1dlop) (natp dgop))
   (cond ((logbitp 1dlop dgop)
           (logxor
             (logand (ash -1 1dlop)
               (ash dgop -1))
             (logandc1 (ash -1 1dlop) dgop)))
         ((and (plusp 1dlop)
              (logbitp (ash (ash -1 1dlop) -1)
                       (ash dgop -1))
              (logandc1 (ash (ash -1 1dlop) -1) dgop)))
         (t (logxor
             (logandc1 (ash -1 1dlop) dgop)
             (logand (ash -1 1dlop)
                      (ash dgop -1))))))
   nil)
   nil))

(defun 1dlop-dgop-indx (1dlop dgop)
 (if (or (logbitp 1dlop dgop)
         (and (plusp 1dlop)
              (logbitp (1- 1dlop) dgop)))
   nil
   (- 1dlop
    (logcount (logandc1 (ash -1 1dlop) dgop))))
Distance from ACL2 code to actual Kenzo code: values

Kenzo

(defun 1dlop-dgop (1dlop dgop)
  (progn
    (when (logbitp 1dlop dgop)
      (let ((share (ash -1 1dlop)))
        (values
          (logxor
            (logand share (ash dgop -1))
            (logandc1 share dgop))
          nil)))
    (when (and (plusp 1dlop)
              (logbitp (1- 1dlop) dgop))
      (let ((share (ash -1 1dlop)))
        (setf share (ash share -1))
        (return-from 1dlop-dgop
          (values
            (logxor
              (logand share (ash dgop -1))
              (logandc1 share dgop))
            nil)))
    (let ((share (ash -1 1dlop)))
      (let ((right (logandc1 share dgop)))
        (values
          (logxor
            right
            (logand share (ash dgop -1)))
          (- 1dlop (logcount right)))))))

ACL2

(defun 1dlop-dgop-dgop (1dlop dgop)
  (if (and (natp 1dlop) (natp dgop))
    (cond ((logbitp 1dlop dgop)
      (logxor
        (logand (ash -1 1dlop)
          (ash dgop -1))
        (logandc1 (ash -1 1dlop)
          dgop)))
      ((and (plusp 1dlop)
        (logbitp (- 1dlop 1) dgop))
        (logxor
          (logand (ash (ash -1 1dlop) -1)
            (ash dgop -1))
          (logandc1 (ash (ash -1 1dlop) -1)
            dgop)))
    (t (logxor
        (logandc1 (ash -1 1dlop) dgop)
        (logand (ash -1 1dlop)
          (ash dgop -1))))))
  nil)

(defun 1dlop-dgop-indx (1dlop dgop)
  (if (or (logbitp 1dlop dgop)
    (and (plusp 1dlop)
      (logbitp (- 1dlop 1) dgop))
    nil
    (- 1dlop (logcount (logandc1 (ash -1 1dlop) dgop))))
  (- (logcount (logandc1 (ash -1 1dlop) dgop)) (- 1dlop)))

J. Heras, V. Pascual, J. Rubio

Proving with ACL2 the correctness of simplicial sets in the Kenzo system
Distance from ACL2 code to actual Kenzo code: loops

Kenzo

(defun cmp-d-ls-dgop (d ls)
 (do ((p ls (cdr p))
      (rsl empty-list (let ((j (car p)))
         (cons (cond ((< d j) (1- j))
                 (t (decf d) j))
        rsl)))
    ((endp p) (nreverse rsl)))
  (when (<= 0 (- d (car p)) 1)
    (return (nreconc rsl (rest p)))))))

ACL2

(defun cmp-d-ls-dgop-do (d p rsl)
  (cond ((endp p) (reverse rsl))
    ((< d (car p))
      (cmp-d-ls-dgop-do d (cdr p)
         (cons (1- (car p)) rsl)))
    ((and (<= 0 (- d (car p)))
          (<= (- d (car p)) 1))
      (append (reverse rsl) (rest p)))
    (t (cmp-d-ls-dgop-do (1- d)
         (cdr p) (cons (car p) rsl))))
  )
)

(defun cmp-d-ls-dgop (d ls)
  (cmp-d-ls-dgop-do d ls nil)
)
Distance from ACL2 code to actual Kenzo code: loops

Kenzo

(defun cmp-d-1s-dgop (d ls)
  (do ((p ls (cdr p)))
      (rsl
        empty-list (let ((j (car p)))
          (cons (cond ((< d j) (1- j))
                  (t (decf d) j))
                  rsl))))
  ((endp p) (nreverse rsl)))
(time (when (<= 0 (- d (car p)) 1)
         (return (nreconc rsl (rest p))))))

ACL2

(defun cmp-d-1s-dgop-do (d p rsl)
  (cond ((endp p) (reverse rsl))
        ((< d (car p))
         (cmp-d-1s-dgop-do d (cdr p)
                            (cons (1- (car p)) rsl))))
        ((and (<= 0 (- d (car p)))
              (<= (- d (car p)) 1))
         (append (reverse rsl) (rest p))
         (t (cmp-d-1s-dgop-do (1- d)
                              (cdr p) (cons (car p) rsl))))
   )
)

(defun cmp-d-1s-dgop (d ls)
  (cmp-d-1s-dgop-do d ls nil))
Generic Simplicial Set Theory

- Framework provides a way of proving that Kenzo Simplicial Sets are really Simplicial Sets.
Generic Simplicial Set Theory

- Framework provides a way of proving that Kenzo Simplicial Sets are really Simplicial Sets
- Automating the proof of Kenzo Simplicial Sets instances
Generic Simplicial Set Theory

- Framework provides a way of proving that Kenzo Simplicial Sets are really Simplicial Sets
- Automating the proof of Kenzo Simplicial Sets instances
  - Generic Instantiation tool
Generic Simplicial Set Theory

- Framework provides a way of proving that Kenzo Simplicial Sets are really Simplicial Sets
- Automating the proof of Kenzo Simplicial Sets instances
  - Generic Instantiation tool

Generic Simplicial Set Theory

- Framework provides a way of proving that Kenzo Simplicial Sets are really Simplicial Sets
- Automating the proof of Kenzo Simplicial Sets instances
  - Generic Instantiation tool
    - Development of generic theories
Generic Simplicial Set Theory

- Framework provides a way of proving that Kenzo Simplicial Sets are really Simplicial Sets
- Automating the proof of Kenzo Simplicial Sets instances
  - Generic Instantiation tool
- Development of generic theories
- Instantiates definitions and theorems of the theory for different instances (different simplicial sets)
Generic Simplicial Set Theory

$\text{encapsulate} \quad \xrightarrow{\text{proof}} \quad \text{Generic Theory}$
Generic Simplicial Set Theory

encapsulate $\rightarrow$ Generic Theory

↑

Instance

From 4 definitions and 4 theorems
Instantiates 3 definitions and 7 theorems
The proof of the 7 theorems involves: 92 definitions and 969 theorems

The proof effort is considerably reduced

J. Heras, V. Pascual, J. Rubio
Generic Simplicial Set Theory

encapsulate \[\rightarrow\] proof \[\rightarrow\] Generic Theory

\[\uparrow\] Instance \[\downarrow\]

Concrete theory

From 4 definitions and 4 theorems
Instantiates 3 definitions and 7 theorems
The proof of the 7 theorems involves: 92 definitions and 969 theorems
The proof effort is considerably reduced
Generic Simplicial Set Theory

\[ \text{encapsulate} \quad \text{proof} \quad \text{Generic Theory} \]

\[ \text{Instance} \quad \downarrow \quad \text{Concrete theory} \]

- Generic Simplicial Set Theory

From 4 definitions and 4 theorems
Instantiates 3 definitions and 7 theorems
The proof of the 7 theorems involves: 92 definitions and 969 theorems
The proof effort is considerably reduced
Generic Simplicial Set Theory

- From 4 definitions and 4 theorems
- From 4 definitions and 4 theorems

\[ \text{Generic Simplicial Set Theory} \]

\[ \text{Generic Theory} \]

\[ \text{Concrete theory} \]

\[ \text{Instance} \]

\[ \text{proof} \]
Generic Simplicial Set Theory

- From 4 definitions and 4 theorems
- Instantiates 3 definitions and 7 theorems
Generic Simplicial Set Theory

- Generic Simplicial Set Theory
  - From 4 definitions and 4 theorems
  - Instantiates 3 definitions and 7 theorems
  - The proof of the 7 theorems involves: 92 definitions and 969 theorems
Generic Simplicial Set Theory

- **Generic Simplicial Set Theory**
  - From 4 definitions and 4 theorems
  - Instantiates 3 definitions and 7 theorems
  - The proof of the 7 theorems involves: 92 definitions and 969 theorems
  - The proof effort is considerably reduced
Certifications of Simplicial Set families

Certification of Kenzo families of simplicial sets:
Certifications of Simplicial Set families

- Certification of Kenzo families of simplicial sets:
  - Spheres: indexed by a natural number
Certifications of Simplicial Set families

- Certification of Kenzo families of simplicial sets:
  - Spheres: indexed by a natural number
  - Simplicial sets coming from simplicial complexes
Certifications of Simplicial Set families

- Certification of Kenzo families of simplicial sets:
  - Spheres: indexed by a natural number
  - Simplicial sets coming from simplicial complexes
  - Standard Simplicial sets: indexed by a natural number
Certifications of Simplicial Set families

- Certification of Kenzo families of simplicial sets:
  - Spheres: indexed by a natural number
  - Simplicial sets coming from simplicial complexes
  - Standard Simplicial sets: indexed by a natural number
- Example: (Standard Simplicial Sets)
Certifications of Simplicial Set families

- Certification of Kenzo families of simplicial sets:
  - Spheres: indexed by a natural number
  - Simplicial sets coming from simplicial complexes
  - Standard Simplicial sets: indexed by a natural number

Example: (Standard Simplicial Sets)

1. Definition of the four functions:

```
(defun face-delta (n i gmsm)
  (cond ((zp i) (cdr gmsm))
        (t (cons (car gmsm) (face-delta n (1- i) (cdr gmsm))))))
(defun dimension-delta (gmsm) ...)
(defun canonical-delta (gmsm) ...)
(defun inv-ss-delta (n gmsm) ...)
```

Proof of the four theorems:

```
(defunthm faceoface-delta
  (implies (and (natp i) (natp j) (< i j) (canonical-delta gmsm))
    (equal (face-delta n i (face-delta n j gmsm))
           (face-delta n (+ -1 j) (face-delta n i gmsm))))
```

J. Heras, V. Pascual, J. Rubio
Proving with ACL2 the correctness of simplicial sets in the Kenzo system
Certifications of Simplicial Set families

- Certification of Kenzo families of simplicial sets:
  - Spheres: indexed by a natural number
  - Simplicial sets coming from simplicial complexes
  - Standard Simplicial sets: indexed by a natural number
- Example: (Standard Simplicial Sets)

1. Definition of the four functions:

```lisp
(defun face-delta (n i gmsm)
  (cond ((zp i) (cdr gmsm))
        (t (cons (car gmsm) (face-delta n (1- i) (cdr gmsm))))))
(defun dimension-delta (gmsm) ...)
(defun canonical-delta (gmsm) ...)
(defun inv-ss-delta (n gmsm) ...)
```

2. Proof of the four theorems:

```lisp
(defthm faceoface-delta
  (implies (and (natp i) (natp j) (< i j) (canonical-delta gmsm))
           (equal (face-delta n i (face-delta n j gmsm))
                  (face-delta n (+ -1 j) (face-delta n i gmsm))))
  ...
)
Certifications of Simplicial Set families

Instantiation of the theory:

```
(defun instance-simplicial-set-kenzo*
  ((face face-delta) (canonical canonical-delta)
   (dimension dimension-delta) (inv-ss inv-ss-delta))
  "-delta")
```

A proof of Kenzo Standard Simplicial Sets are really Simplicial Sets is automatically generated.
Certifications of Simplicial Set families

3 Instantiation of the theory:

(definstance-*simplicial-set-kenzo*
  ((face face-delta) (canonical canonical-delta)
   (dimension dimension-delta) (inv-ss inv-ss-delta))
  "-delta")

4 A proof of Kenzo Standard Simplicial Sets are really Simplicial Sets is automatically generated
Table of Contents
Conclusions

Conclusions:

- Framework to prove the correctness of Kenzo simplicial sets
- Proof of the correctness of families of simplicial sets
- Considerable reduction of the proof effort

Methodology for Kenzo constant spaces constructors

J. Heras, V. Pascual, J. Rubio
Conclusions

- Conclusions:
  - Framework to prove the correctness of Kenzo simplicial sets
Conclusions

Conclusions:
- Framework to prove the correctness of Kenzo simplicial sets
- Proof of the correctness of families of simplicial sets
Conclusions

Conclusions:

- Framework to prove the correctness of Kenzo simplicial sets
- Proof of the correctness of families of simplicial sets
- Considerable reduction of the proof effort
Conclusions:

- Framework to prove the correctness of Kenzo simplicial sets
  - Proof of the correctness of families of simplicial sets
  - Considerable reduction of the proof effort
- Methodology for Kenzo constant spaces constructors
Further Work:

- Prove the correctness of other Kenzo simplicial sets
- Moore spaces
- Eilenberg-MacLane spaces
- Apply the presented methodology to other Kenzo data structures which model mathematical structures
- Certify the constructors of new spaces from other ones
- Higher-order functional programming is involved
- Automating the transformations between Kenzo and ACL2
Further Work:

- Prove the correctness of other Kenzo simplicial sets
  - Moore spaces
  - Eilenberg-MacLane spaces
  - ...
Further Work:

- Prove the correctness of other Kenzo simplicial sets
  - Moore spaces
  - Eilenberg-MacLane spaces
  - ...

- Apply the presented methodology to other Kenzo data structures which model mathematical structures
Further Work

- Prove the correctness of other Kenzo simplicial sets
  - Moore spaces
  - Eilenberg-MacLane spaces
  - ...
- Apply the presented methodology to other Kenzo data structures which model mathematical structures
- Certify the constructors
  - construction of new spaces from other ones
  - higher-order functional programming is involved
Further Work:

- Prove the correctness of other Kenzo simplicial sets
  - Moore spaces
  - Eilenberg-MacLane spaces
  - ... 
- Apply the presented methodology to other Kenzo data structures which model mathematical structures
- Certify the constructors
  - construction of new spaces from other ones
  - higher-order functional programming is involved
- Automating the transformations between Kenzo and ACL2
Proving with ACL2 the correctness of simplicial sets in the Kenzo system

Jónathan Heras    Vico Pascual    Julio Rubio

Departamento de Matemáticas y Computación
Universidad de La Rioja
Spain

20th International Symposium on Logic-Based Program Synthesis and Transformation
LOPSTR 2010, Hagenberg, Austria