Proving with ACL2 the correctness of simplicial sets in the Kenzo system[1]

Jónathan Heras   Vico Pascual   Julio Rubio

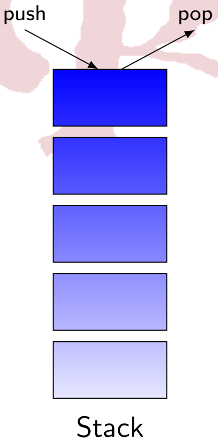
Departamento de Matemáticas y Computación
Universidad de La Rioja
Spain

20th International Symposium on Logic-Based Program Synthesis and Transformation

LOPSTR 2010, Hagenberg, Austria

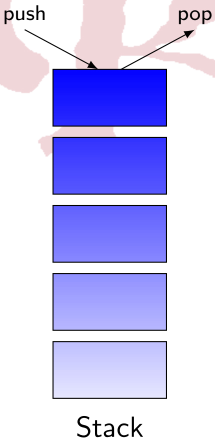
[1] Partially supported by Ministerio de Educación y Ciencia, project MTM2009-13842-C02-01, and by European Commission FP7, STREP project ForMath

Introductory Example



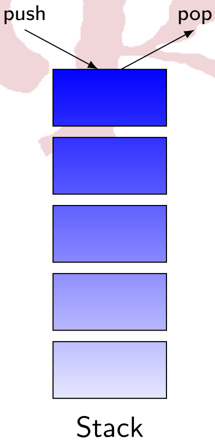
- Implementation of stacks

Introductory Example



- Implementation of stacks
- Prove the correctness of our implementation

Introductory Example



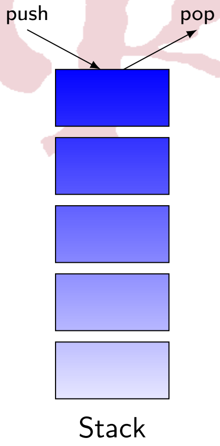
- Implementation of stacks
- Prove the correctness of our implementation
  - Model the problem

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

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

(defun pop (stack)
  (cdr stack))
```

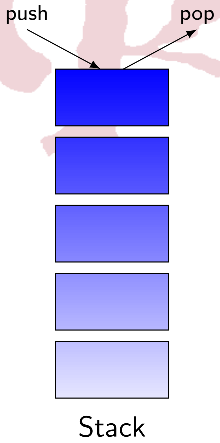
Introductory Example



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

```
(defthm push-pop
  (implies (stack-p stack)
           (equal (pop (push a stack))
                  stack)))
...
```

Introductory Example



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

$\Rightarrow$ Our implementation of a stack is really a stack

```
(defthm push-pop
  (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:

Current Work

- Kenzo way of working:
  1. 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,...): $\sim 60\%$

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 $\rightarrow$ ACL2

Case Study

Each Kenzo Simplicial Set is really a simplicial set

Table of Contents

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 disjoints sets, together with functions:

$$\begin{array}{lll} \partial_i^q : K^q \to K^{q-1}, & q > 0, & i = 0, \ldots, q, \\ \eta_i^q : K^q \to K^{q+1}, & q \geq 0, & i = 0, \ldots, q, \end{array}$$

subject to the relations:

$$\begin{array}{llllll} (1) & \partial_i^{q-1} \partial_j^q & = & \partial_{j-1}^{q-1} \partial_i^q & \text{if} & i < j, \\ (2) & \eta_i^{q+1} \eta_j^q & = & \eta_{j+1}^{q+1} \eta_i^q & \text{if} & i \leq j, \\ (3) & \partial_i^{q+1} \eta_j^q & = & \eta_{j-1}^{q-1} \partial_i^q & \text{if} & i < j, \\ (4) & \partial_i^{q+1} \eta_i^q & = & \mathit{identity} & = & \partial_{i+1}^{q+1} \eta_i^q, \\ (5) & \partial_i^{q+1} \eta_j^q & = & \eta_j^{q-1} \partial_{i-1}^q & \text{if} & i > j+1, \end{array}$$

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:

$$\begin{array}{lll} \partial_i^q : K^q \to K^{q-1}, & q > 0, & i = 0, \ldots, q, \\ \eta_i^q : K^q \to K^{q+1}, & q \geq 0, & i = 0, \ldots, q, \end{array}$$

subject to the relations:

$$\begin{array}{llllll} (1) & \partial_i^{q-1}\partial_j^q & = & \partial_{j-1}^{q-1}\partial_i^q & \text{if} & i < j, \\ (2) & \eta_i^{q+1}\eta_j^q & = & \eta_{j+1}^{q+1}\eta_i^q & \text{if} & i \leq j, \\ (3) & \partial_i^{q+1}\eta_j^q & = & \eta_{j-1}^{q-1}\partial_i^q & \text{if} & i < j, \\ (4) & \partial_i^{q+1}\eta_i^q & = & \textit{identity} & = & \partial_{i+1}^{q+1}\eta_i^q, \\ (5) & \partial_i^{q+1}\eta_j^q & = & \eta_j^{q-1}\partial_{i-1}^q & \text{if} & i > j+1, \end{array}$$

- The elements of K^q are called q-simplexes

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:

$$\begin{array}{lll} \partial_i^q : K^q \to K^{q-1}, & q > 0, & i = 0, \dots, q, \\ \eta_i^q : K^q \to K^{q+1}, & q \geq 0, & i = 0, \dots, q, \end{array}$$

subject to the relations:

$$\begin{array}{llllll} (1) & \partial_i^{q-1}\partial_j^q & = & \partial_{j-1}^{q-1}\partial_i^q & \text{if} & i < j, \\ (2) & \eta_i^{q+1}\eta_j^q & = & \eta_{j+1}^{q+1}\eta_i^q & \text{if} & i \leq j, \\ (3) & \partial_i^{q+1}\eta_j^q & = & \eta_{j-1}^{q-1}\partial_i^q & \text{if} & i < j, \\ (4) & \partial_i^{q+1}\eta_i^q & = & \mathit{identity} & = & \partial_{i+1}^{q+1}\eta_i^q, \\ (5) & \partial_i^{q+1}\eta_j^q & = & \eta_j^{q-1}\partial_{i-1}^q & \text{if} & i > j+1, \end{array}$$

- 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:

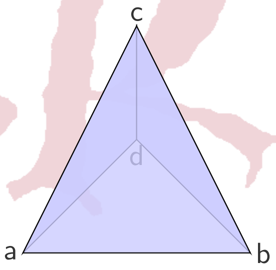
$$\begin{array}{lll} \partial_i^q : K^q \to K^{q-1}, & q > 0, & i = 0, \ldots, q, \\ \eta_i^q : K^q \to K^{q+1}, & q \geq 0, & i = 0, \ldots, q, \end{array}$$

subject to the relations:

$$\begin{array}{llllll} (1) & \partial_i^{q-1} \partial_j^q & = & \partial_{j-1}^{q-1} \partial_i^q & \text{if} & i < j, \\ (2) & \eta_i^{q+1} \eta_j^q & = & \eta_{j+1}^{q+1} \eta_i^q & \text{if} & i \leq j, \\ (3) & \partial_i^{q+1} \eta_j^q & = & \eta_{j-1}^{q-1} \partial_i^q & \text{if} & i < j, \\ (4) & \partial_i^{q+1} \eta_i^q & = & \textit{identity} & = & \partial_{i+1}^{q+1} \eta_i^q, \\ (5) & \partial_i^{q+1} \eta_j^q & = & \eta_j^{q-1} \partial_{i-1}^q & \text{if} & i > j+1, \end{array}$$

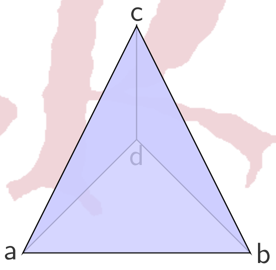
- The elements of K^q are called q-simplexes
- A q-simplex x is degenerate if $x = \eta_i^{q-1} y$ for some simplex $y \in K^{q-1}$
- Otherwise x is called non-degenerate

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: $(a\ b\ c\ d)$

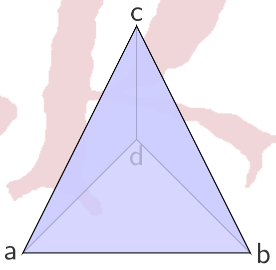
Mathematical context: Example



- 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: $(a\ b\ c\ d)$

$$\text{face: } \partial_i(a\ b\ c) = \left\{ \begin{array}{ll} (b\ c) & \text{if } i = 0 \\ (a\ c) & \text{if } i = 1 \\ (a\ b) & \text{if } i = 2 \end{array} \right\} \text{geometrical meaning}$$

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: $(a\ b\ c\ d)$

$$\text{face: } \partial_i(a\ b\ c) = \left\{ \begin{array}{ll} (b\ c) & \text{if } i = 0 \\ (a\ c) & \text{if } i = 1 \\ (a\ b) & \text{if } i = 2 \end{array} \right\} \text{geometrical meaning}$$

$$\text{degeneracy: } \eta_i(a\ b\ c) = \left\{ \begin{array}{ll} (a\ a\ b\ c) & \text{if } i = 0 \\ (a\ b\ b\ c) & \text{if } i = 1 \\ (a\ b\ c\ c) & \text{if } i = 2 \end{array} \right\} \text{non-geometrical meaning}$$

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:

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

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:

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

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:

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

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

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

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:

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

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

- *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:

	simplex	abstract simplex
non-degenerate	$(a \; b)$	$(\emptyset \; (a \; b))$

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:

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

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

- *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:

	simplex	abstract simplex
non-degenerate	$(a \; b)$	$(\emptyset \; (a \; b))$
degenerate	$(a \; a \; b \; c)$	$(\eta_0 \; (a \; b \; c))$

Mathematical context: face and degeneracy

- *degeneracy operator*: $\eta_i^q(dgop \quad gmsm) := (\eta_i^q \circ dgop \quad gmsm)$

Mathematical context: face and degeneracy

- *degeneracy operator*: $\eta_i^q(dgop \quad gmsm) := (\eta_i^q \circ dgop \quad gmsm)$
  - Independent from the simplicial set

Mathematical context: face and degeneracy

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

Mathematical context: face and degeneracy

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

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

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

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

where

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

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

  - Dependent from the simplicial set ...

Mathematical context: face and degeneracy

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

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

where

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

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

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

Mathematical context: face and degeneracy

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

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

where

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

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

  - Dependent from the simplicial set ...
  - but some parts are independent
  - $\partial_2(\eta_3\eta_1\ (a\ b\ c)) = (\partial_2\eta_3\eta_1\ (a\ b\ c)) \underset{\partial_{i+1}\eta_i=identity}{\overset{\partial_i\eta_j=\eta_{j-1}\partial_i \text{ if } i<j}{=}} (\eta_2\ (a\ b\ c))$

Mathematical context: face and degeneracy

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

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

where

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

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

  - Dependent from the simplicial set ...
  - but some parts are independent
  - $\partial_2(\eta_3\eta_1\ (a\ b\ c)) = (\partial_2\eta_3\eta_1\ (a\ b\ c)) \underset{\partial_{i+1}\eta_i=identity}{\overset{\partial_i\eta_j=\eta_{j-1}\partial_i \text{ if } i<j}{=}} (\eta_2\ (a\ b\ c))$
  - $\partial_2(\eta_3\eta_0\ (a\ b\ c)) = (\partial_2\eta_3\eta_0\ \partial_1(a\ b\ c)) \underset{\partial_i\eta_j = \eta_j\partial_{i-1} \text{ if } i>j+1}{\overset{\partial_i\eta_j=\eta_{j-1}\partial_i \text{ if } i<j}{=}} (\eta_2\eta_0\ (a\ c))$

Mathematical context: minimal conditions

Theorem

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

1. $\forall i, j \in \mathbb{N} : i < j \leq q \Longrightarrow \widehat{\partial}_i^{q-1}(\widehat{\partial}_j^q gmsm) = \widehat{\partial}_{j-1}^{q-1}(\widehat{\partial}_i^q gmsm)$,
2. $\forall i \in \mathbb{N}$, $i \leq q$: $\widehat{\partial}_i^q gmsm \in K^{q-1}$,

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

ACL2 framework: minimal conditions

Theorem

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

1. $\forall i, j \in \mathbb{N} : i < j \leq q \Longrightarrow \widehat{\partial}_i^{q-1}(\widehat{\partial}_j^q gmsm) = \widehat{\partial}_{j-1}^{q-1}(\widehat{\partial}_i^q gmsm)$,
2. $\forall i \in \mathbb{N}$, $i \leq q$: $\widehat{\partial}_i^q gmsm \in K^{q-1}$,

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

ACL2 framework: minimal conditions

Theorem

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

1. $\forall i,j \in \mathbb{N} : i < j \leq q \Longrightarrow \widehat{\partial}_i^{q-1}(\widehat{\partial}_j^q gmsm) = \widehat{\partial}_{j-1}^{q-1}(\widehat{\partial}_i^q gmsm)$,
2. $\forall i \in \mathbb{N}, i \leq q$: $\widehat{\partial}_i^q gmsm \in K^{q-1}$,

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

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

ACL2 framework: minimal conditions

Theorem

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

1. $\forall i, j \in \mathbb{N} : i < j \leq q \Longrightarrow \widehat{\partial}_i^{q-1}(\widehat{\partial}_j^q gmsm) = \widehat{\partial}_{j-1}^{q-1}(\widehat{\partial}_i^q gmsm)$,
2. $\forall i \in \mathbb{N}, i \leq q$: $\widehat{\partial}_i^q gmsm \in K^{q-1}$,

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 $\{K^q, \widehat{\partial}^q\}_{q\geq 0}$ *such that for all element gmsm* $\in K^q$ *the following properties hold:*

1. $\forall i,j \in \mathbb{N} : i < j \leq q \Longrightarrow \widehat{\partial}_i^{q-1}(\widehat{\partial}_j^q gmsm) = \widehat{\partial}_{j-1}^{q-1}(\widehat{\partial}_i^q gmsm)$,
2. $\forall i \in \mathbb{N},\ i \leq q$: $\widehat{\partial}_i^q gmsm \in K^{q-1}$,

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)))))
(defthm inv-ss-prop
  (implies (and (canonical absm) (natp i) (< i (dimension absm)))
  (equal (dimension (face ss i absm)) (1- (dimension absm)))
; Witness ... )
```

ACL2 framework: face and degeneracy

Theorem

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

1. $\forall i, j \in \mathbb{N} : i < j \leq q \Longrightarrow \widehat{\partial}_i^{q-1}(\widehat{\partial}_j^q gmsm) = \widehat{\partial}_{j-1}^{q-1}(\widehat{\partial}_i^q gmsm)$,
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

```
(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

Theorem

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

1. $\forall i, j \in \mathbb{N} : i < j \leq q \Longrightarrow \partial_i^{q-1}(\partial_j^q gmsm) = \partial_{j-1}^{q-1}(\partial_i^q gmsm)$,
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 $\{K^q, \widehat{\partial}^q\}_{q\geq 0}$ such that for all element gmsm $\in K^q$ the following properties hold:

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

- imp-face-Kenzo and imp-degeneracy-Kenzo satisfy the 5 properties of simplicial sets

```
(defthm theorem-3
(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))))))
```

Sketch of the proofs

Methodological approach imported from:

F. J. Martín-Mateos, J. Rubio, and J. L. Ruiz-Reina. ACL2 verification of simplical degeneracy programs in the Kenzo system. Lecture Notes in Computer Science, 5625:106–121, 2009.

Sketch of the proofs

Methodological approach imported from:

F. J. Martín-Mateos, J. Rubio, and J. L. Ruiz-Reina. ACL2 verification of simplical degeneracy programs in the Kenzo system. Lecture Notes in Computer Science, 5625:106–121, 2009.

1. Prove each theorem with EAT representation

Sketch of the proofs

Methodological approach imported from:

F. J. Martín-Mateos, J. Rubio, and J. L. Ruiz-Reina. ACL2 verification of simplical degeneracy programs in the Kenzo system. Lecture Notes in Computer Science, 5625:106–121, 2009.

1. Prove each theorem with EAT representation
   - EAT is the predecessor of Kenzo

Sketch of the proofs

Methodological approach imported from:

F. J. Martín-Mateos, J. Rubio, and J. L. Ruiz-Reina. ACL2 verification of simplical degeneracy programs in the Kenzo system. Lecture Notes in Computer Science, 5625:106–121, 2009.

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:

F. J. Martín-Mateos, J. Rubio, and J. L. Ruiz-Reina. ACL2 verification of simplical degeneracy programs in the Kenzo system. Lecture Notes in Computer Science, 5625:106–121, 2009.

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:

F. J. Martín-Mateos, J. Rubio, and J. L. Ruiz-Reina. ACL2 verification of simplical degeneracy programs in the Kenzo system. Lecture Notes in Computer Science, 5625:106–121, 2009.

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

`imp-face-eat`	$\Leftrightarrow$	`imp-face-Kenzo`
`imp-degeneracy-eat`	$\Leftrightarrow$	`imp-degeneracy-Kenzo`
`imp-inv-eat`	$\Leftrightarrow$	`imp-inv-Kenzo`

Sketch of the proofs

Methodological approach imported from:

F. J. Martín-Mateos, J. Rubio, and J. L. Ruiz-Reina. ACL2 verification of simplical degeneracy programs in the Kenzo system. Lecture Notes in Computer Science, 5625:106–121, 2009.

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

`imp-face-eat`	$\Leftrightarrow$	`imp-face-Kenzo`
`imp-degeneracy-eat`	$\Leftrightarrow$	`imp-degeneracy-Kenzo`
`imp-inv-eat`	$\Leftrightarrow$	`imp-inv-Kenzo`

$\Rightarrow$ All the theorems are proved with Kenzo representation

EAT/Kenzo representation

EAT

Kenzo

EAT/Kenzo representation

EAT

- abstract simplexes:

  (*dgop* *gmsm*) :=

  $\begin{cases} \textit{dgop} \text{ is a strictly decreasing list} \\ \textit{gmsm} \text{ is an object} \end{cases}$

  Example:

  $(\eta_3\eta_1\ (a\ b\ c)) \rightsquigarrow ((3\ 1)\ (a\ b\ c))$

Kenzo

- abstract simplexes:

  (*dgop* *gmsm*) :=

  $\begin{cases} \textit{dgop} \text{ is a natural number} \\ \textit{gmsm} \text{ is an object} \end{cases}$

  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*) :=

  $\begin{cases} \textit{dgop} \text{ is a strictly decreasing list} \\ \textit{gmsm} \text{ is an object} \end{cases}$

  Example:

  $(\eta_3\eta_1\ (a\ b\ c)) \rightsquigarrow ((3\ 1)\ (a\ b\ c))$

- face, degeneracy:
  implemented with recursive functions

Kenzo

- abstract simplexes:

  (*dgop* *gmsm*) :=

  $\begin{cases} \textit{dgop} \text{ is a natural number} \\ \textit{gmsm} \text{ is an object} \end{cases}$

  Example:
  $(\eta_3\eta_1\ (a\ b\ c)) \rightsquigarrow (10\ (a\ b\ c))$

  $\eta_3\eta_1 \rightsquigarrow (0\ 1\ 0\ 1) \rightsquigarrow$

  $0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 = 10$

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

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:

  $(\eta_3\eta_1 \; (a \; b \; c)) \rightsquigarrow ((3\;1) \; (a \; b \; c))$

- face, degeneracy:
  implemented with recursive functions

- inefficient

- easy to prove

Kenzo

- abstract simplexes:

  $(dgop \;\; gmsm) :=$

  $\begin{cases} dgop \text{ is a natural number} \\ gmsm \text{ is an object} \end{cases}$

  Example:
  $(\eta_3\eta_1 \; (a \; b \; c)) \rightsquigarrow (10 \; (a \; b \; c))$

  $\eta_3\eta_1 \rightsquigarrow (0\;1\;0\;1) \rightsquigarrow$

  $0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 = 10$

- 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

```
(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))))))
```

1. First we prove

```
(defthm theorem-3-eat
(implies (and (imp-inv-eat ss q (dgop gmsm)) (natp i) (natp j) (< i j))
         (equal (imp-face-eat ss i (1- q) (imp-face-eat ss j q (dgop gmsm)))
                (imp-face-eat ss (1- j) (1- q) (imp-face-eat ss i q (dgop gmsm))))))
```

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))))))
```

1. First we prove

```
(defthm theorem-3-eat
(implies (and (imp-inv-eat ss q (dgop gmsm)) (natp i) (natp j) (< i j))
         (equal (imp-face-eat ss i (1- q) (imp-face-eat ss j q (dgop gmsm)))
                (imp-face-eat ss (1- j) (1- q) (imp-face-eat ss i q (dgop gmsm))))))
```

   - induction
   - simplification
   - study of cases

Proof of a theorem continued

2. then we prove

   `imp-face-eat` $\Leftrightarrow$ `imp-face-Kenzo`

Proof of a theorem continued

2. then we prove

   `imp-face-eat` $\Leftrightarrow$ `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

   `imp-face-eat` $\Leftrightarrow$ `imp-face-Kenzo`

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

Proof of a theorem continued

2. then we prove

`imp-face-eat` $\Leftrightarrow$ `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

mathematical	EAT	Binary	Kenzo
$\eta_3\eta_1$	(3 1)	(0 1 0 1)	10

Proof of a theorem continued

2. then we prove

   `imp-face-eat` $\Leftrightarrow$ `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

mathematical	EAT	Binary	Kenzo
$\eta_3\eta_1$	(3 1)	(0 1 0 1)	10

- Definition of `imp-face-binary`
  - Works with binary lists
  - Inspired from Kenzo functions

Proof of a theorem continued

2. then we prove

`imp-face-eat` $\Leftrightarrow$ `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

mathematical	EAT	Binary	Kenzo
$\eta_3\eta_1$	(3 1)	(0 1 0 1)	10

- Definition of `imp-face-binary`
  - Works with binary lists
  - Inspired from Kenzo functions

`imp-face-eat` $\Leftrightarrow$ `imp-face-binary` $\Leftrightarrow$ `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 (- 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)
```

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)
```

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-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)
  )
```

Table of Contents

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
    - F. J. Martín-Mateos, J. A. Alonso, M. J. Hidalgo, and J. L. Ruiz-Reina. A Generic Instantiation Tool and a Case Study: A Generic Multiset Theory. Proceedings of the Third ACL2 workshop. Grenoble, Francia, pp. 188–203, 2002.

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
    - F. J. Martín-Mateos, J. A. Alonso, M. J. Hidalgo, and J. L. Ruiz-Reina. A Generic Instantiation Tool and a Case Study: A Generic Multiset Theory. Proceedings of the Third ACL2 workshop. Grenoble, Francia, pp. 188–203, 2002.
    - 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
    - F. J. Martín-Mateos, J. A. Alonso, M. J. Hidalgo, and J. L. Ruiz-Reina. A Generic Instantiation Tool and a Case Study: A Generic Multiset Theory. Proceedings of the Third ACL2 workshop. Grenoble, Francia, pp. 188–203, 2002.
    - Development of generic theories
    - Instantiates definitions and theorems of the theory for different instances (different simplicial sets)

Generic Simplicial Set Theory

$$\text{encapsulate} \xrightarrow{\textit{proof}} \text{Generic Theory}$$

Generic Simplicial Set Theory

$$\begin{array}{ccc} \text{encapsulate} & \xrightarrow{\textit{proof}} & \text{Generic Theory} \\ \uparrow & & \\ \text{Instance} & & \end{array}$$

Generic Simplicial Set Theory

$$\begin{array}{ccc} \text{encapsulate} & \xrightarrow{\textit{proof}} & \text{Generic Theory} \\ \uparrow & & \downarrow \\ \text{Instance} & & \text{Concrete theory} \end{array}$$

Generic Simplicial Set Theory

$$\begin{array}{ccc} \text{encapsulate} & \xrightarrow{\textit{proof}} & \text{Generic Theory} \\ \uparrow & & \downarrow \\ \text{Instance} & & \text{Concrete theory} \end{array}$$

- Generic Simplicial Set Theory

Generic Simplicial Set Theory

$$
\begin{array}{ccc}
\text{encapsulate} & \xrightarrow{\textit{proof}} & \text{Generic Theory} \\
\uparrow & & \downarrow \\
\text{Instance} & & \text{Concrete theory}
\end{array}
$$

- Generic Simplicial Set Theory
  - From 4 definitions and 4 theorems

Generic Simplicial Set Theory

$$\begin{array}{ccc} \text{encapsulate} & \xrightarrow{\textit{proof}} & \text{Generic Theory} \\ \uparrow & & \downarrow \\ \text{Instance} & & \text{Concrete theory} \end{array}$$

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

Generic Simplicial Set Theory

$$\begin{array}{ccc} \text{encapsulate} & \xrightarrow{\textit{proof}} & \text{Generic Theory} \\ \uparrow & & \downarrow \\ \text{Instance} & & \text{Concrete theory} \end{array}$$

- 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

$$\begin{array}{ccc} \text{encapsulate} & \xrightarrow{\textit{proof}} & \text{Generic Theory} \\ \uparrow & & \downarrow \\ \text{Instance} & & \text{Concrete theory} \end{array}$$

- 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) ...)
```

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) ...)
```

  2. Proof of the four theorems:

```
(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

3. Instantiation of the theory:

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

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:

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

- 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

- Further Work:

Further Work

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

Further Work

- 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

- 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

- 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