# Using the *Larch Prover* to Derive Relationships for Cubical Complexes

**by:** **Robert Ellis**
**for:** **CS6104, Symbolic Computation**
**date:** **June 27, 1996**

## Abstract

The *Larch Prover* is a symbolic logic system for axiomatizing and proving theorems and investigating rewrite systems in general (e.g., term and string rewriting systems). To be investigated in this project are the proofs of to important inequalities in the study of cubical complexes which lie in cubes. A modularized proof is presented in *Larch* which encodes the relations necessary to prove the inequalities and gives the symbolic flow of the proof.

## 1 Introduction to *Larch*

Based on the course content of CS6104, the aspects of *Larch* used in this project are best seen as a term rewriting system with a few additional rules about how variables are viewed and how new rewrite rules are obtained. Here is a list of components of *Larch* used in the proof of the inequalities for cubical complexes.

1. **Sorts.** These are classes of data; for instance, natural numbers, real numbers, and sets are usually declared as sorts. Sorts can be viewed as classes of variables in a rewrite system. Classifying variables streamlines the reduction processes by allowing for type-checking.

2. **Operators.** Operators are analogous to functions in rewriting systems. Operators take zero or more sorts as arguments and return a sort. For

example, the successor function for natural numbers is an operator. Infix operators may also be declared (or overloaded).

3. **Generation Rules.** The natural numbers are generated by zero and the successor function. Information like this is encoded as a generation rule, or induction rule, that can be used in an inductive proof on a variable that is of a sort which has a generation rule defined.

4. **Variables.** These correspond to variables in term rewriting systems; however, variables are associated with a specific sort, and generation rules and assertions are type-checked accordingly.

5. **Assertions.** Facts necessary for the proof are written as some sort of expression which evaluates to *true*. These might be axioms of the relevant topic or theorems that are assumed for the sake of argument simplification. All of the assertions together give a list of legal rewrite rules that can be used to reduce other expressions to true (e.g., in proving theorems).

6. **Proofs.** A proof is an attempt to reduce an expression to *true*. This can be done by application of rewrite rules for asserted expressions, or by one of a number of proof methods, including induction, proof by cases, and proof by implication or contradiction.

In general, a term rewriting system does not give the same information as the equational system from which it was generated. The discrepancy most often appears as an expression reducing to *true* in the equational system, but not in the rewriting system. Since finding the complete set of true expressions in an equational system is often an *NP-hard* problem, corresponding expressions of the rewrite system might not be completely reducible to *true* or *false* by the initial set of rules. *Larch* provides a deliberate, systematic method of extending the term rewriting system to evaluate more expressions that can be evaluated by the equational system (of course, if an expression has an associated boolean value in both the rewriting system and the equational system, the values must be the same). The goal of *Larch* is, given a particular expression to evaluate, to convert equations of the system into rewrite rules that can be applied to the expression in order to effect a reduction. In the case of proof by induction, *Larch* provides a construct in which the expression reduces to *true* if *Larch* is able to reduce the basis step to *true* and then reduce the inductive step to *true*, using the inductive hypothesis as a temporary assertion.

# 2 *Larch* and this Project

The original goal of this project was to achieve the proof of two inequalities of cubical complexes by building up all theory from the axiomatization of set theory, natural numbers, and definition of two operators on cubical complexes. It soon became evident that working from scratch would be overambitious and time-consuming. Instead, certain "theorems" were stated as axioms, such as the theorem

$$X \subseteq Y \Rightarrow |X| \leq |Y| \quad \text{for sets } X \text{ and } Y,$$

and others which are generally accepted as true in mathematical fields. Derivation of some of these theorems would have required extra work in directions that were deemed nonessential to the core of the application, as these types of theorems have been proved before (in *Larch* and otherwise). Asserting these kinds of theorems focuses attention on the work being done in the proofs of the two inequalities on cubical complexes rather than the tedious details of the generative rules for natural numbers, set theory, and set cardinality.

## 3 Introduction to Cubical Complexes

**Definition.** *A cubical complex $C$ (which lies in a cube) is a set of faces of an $n$-dimensional cube with the property that if $F \in C$ is a face of dimension $r$, where $0 \leq r \leq n$, then all lower dimensional faces contained in $F$ are also in $C$.*

A standard visualization of a cubical complex is a set of vertices, edges, squares, cubes, and higher dimensional faces sitting in a binary lattice of dimension $n$. A face in the cubical complex is labelled by the lowest and highest vertex in the face. Since the visualization resides in a binary lattice, it is possible to associate each vertex with a $n$-digit binary word. This allows the standard partial order on binary words (where binary words represent sets and the order is by inclusion) to be applied to vertices; thus in cubical faces all vertices of a face are ancestors of the last vertex of the face and descendants of the first vertex in the face. Figure 1 displays an example of a cubical complex.

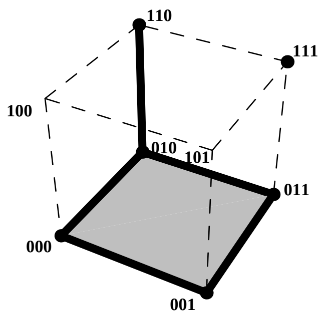


Figure 1: A cubical complex in a 3-dimensional lattice.

There is a total order on all faces of the cubical complex defined from the partial order. It is not important to *Larch* what this total order is; this total order leads to the development of a theorem that will be asserted in order to obtain the desired inequalities. Based on this total order the replacement operator $R$ is defined.

**Definition.** *If $S_r$ is a set of $r$-dimensional faces of the $n$-cube, then $RS_r$ is defined to be the first $|S_r|$ $r$-faces in the total order of faces in the $n$-cube, and the operator $R$ is called the replacement operator.*

It is not hard to see that, given two sets of $r$-faces $S_r$ and $S'_r$, the following implication holds:

$$S_r \subseteq S'_r \Rightarrow RS_r \subseteq RS'_r.$$

This will be asserted in *Larch* for the proof of the inequalities.

By the definition of a cubical complex, all the faces contained in a face of the complex must also be in the complex. From this it is evident that the idea of *boundary* of a face is important. The boundary of a face is now defined.

**Definition.** *Let $S_r$ be a set of $r$-faces. The boundary set $\partial S_r$ is the set of $(r-1)$-faces that are subfaces of $r$-faces of $S_r$ The coboundary set $\partial^{-1} S_r$ is the largest set of $(r+1)$-faces $S'$ for which $\partial S' \subseteq \partial S_r$.*

The idea of boundary is illustrated in Figure 2.

The 0th boundary (or coboundary) of a set $S_r$, or $\partial^0 S_r$, is defined to be $S_r$ itself. Composition of boundary operation is allowed, as is composition of coboundary operation. It is not hard to further see that the $\partial$ and $\partial^{-1}$ respect set inclusion. Finally, it is not difficult to see that $\partial^{-1}\partial S_r = S_r$. This gives the following rules to be assumed in the proof of the inequalities

$(i \geq 0)$:

$$\begin{aligned}
\partial^0 S_r &= S_r \\
\partial \partial^i S_r &= \partial^{i+1} S_r \\
\partial^{-1} \partial^{-i} S_r &= \partial^{-i-1} S_r \\
S_r \subseteq S'_r &\Rightarrow \partial S_r \subseteq \partial S'_r \\
S_r \subseteq S'_r &\Rightarrow \partial^{-1} S_r \subseteq \partial^{-1} S'_r \\
\partial^{-1} \partial S_r &= S_r
\end{aligned}$$

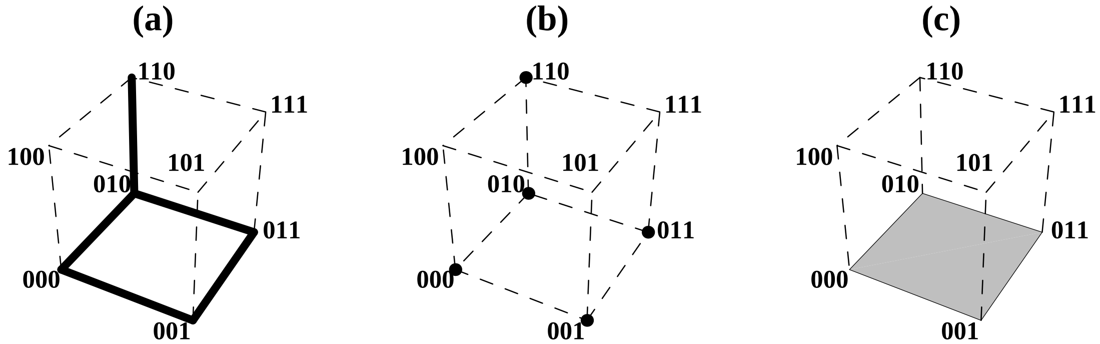


Figure 2: A set of 1-faces appears in (a), with its boundary in (b) and its coboundary in (c).

## 3.1 The Crucial Inclusion

These definitions allow the phrasing of the main theorem that will be assumed in the proof of the two inequalities for cubical complexes.

**Theorem (The Crucial Inclusion).** *Let $S_r$ be a set of $r$-faces of the $n$-cube, and let $\partial$ and $R$ be the operators defined previously. Then*

$$\partial R S_r \subseteq R \partial S_r.$$

Essentially, the boundary operator $\partial$ can be moved past the replacement operator $R$, as long as set inclusion is maintained in the reduction. The proof of this theorem appears in [3] (a corrected version appears in [1]).

## 3.2 $f$-Vectors of Cubical Complexes

The $f$-vector of a cubical complex encodes in a tuple the number of faces of each dimension in the complex. Of significant interest in the study of cubical complexes is the complete characterization of the $f$-vectors of all legal cubical complexes. The characterization question is still open for arbitrary cubical complexes, but it has been solved for cubical complexes which lie in cubes or binary lattices (the characterization appears in [2] and is extended in [1]).

The *crucial inclusion* above is the main theorem needed for the characterization; in particular, there are two inequalities that follow as a corollary of the crucial inclusion. The inequalities are of a more computational nature, and lead more directly to the proof of the characterization. It is this corollary that will be proved by use of *Larch*, with the assumption of the crucial inclusion.

## 3.3 Corollary to the Crucial Inclusion

**Corollary to the Crucial Inclusion.** *Let $S_r$ be a set of $r$-faces of the $n$-cube, and let $\partial$, $\partial$, and $R$ be the boundary, coboundary, and replacement operators, respectively. Then*

$$\begin{aligned} |\partial^p R S_r| &\leq |\partial^p S_r| \\ |\partial^{-p} R S_r| &\geq |\partial^{-p} S_r|, \end{aligned}$$

*where* $0 \leq p$.

An illustration of the first inequality appears in Figure 3.

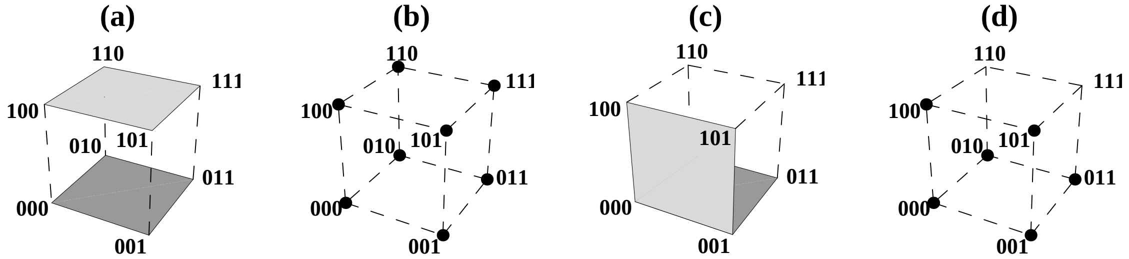

Figure 3: A set of 2-faces $S_2$ appears in (a). The replaced set $RS_2$ appears in (c). $\partial^2 S_2$ and $\partial^2 RS_2$ appear in (b) and (d), respectively.

The strategy for proving of this corollary in *Larch* will be to assume the crucial inclusion and the given facts about the operators and use a minimal number of generative axioms for set theory, natural numbers, and cardinality. These facts and axioms will be converted into rewrite rules in a form that *Larch* can understand, and then the proof methods of *Larch* will be used to prove the inequalities by induction on $p$.

# 4 Executing the Proof in *Larch*

Now that all of the operators are defined and it has been decided what facts and theorems to assert as true, it remains to cajole *Larch* into proving the inequalities of the corollary by techniques of term rewriting systems. The techniques used primarily in the proof of the corollary are normalization, unification, instantiation, critical pairs, and induction.

Normalization and unification are methods of rewriting a subterm of one expression using another expression, with the goal of reducing the first expression to a more manageable form. Instantiation involves replacing a variable in an expression with a term, in order to further normalize or otherwise reduce the result. Finding critical pairs is a pattern-matching scheme by which two terms are analyzed for overlap, and resultant non-resolving term pairs are added to the list of rewrite rules. Often finding critical pairs can be viewed as a generalized form of instantiation. Induction is a method by which an infinite set of rules is represented. The basis step must be proved (reduced to *true*) before the induction hypothesis may be assumed in order to use it, the other assertions and rewrite rules, and whatever proof method in order to prove the inductive step.

## Proof of the First Inequality

In order to show that $|\partial^i RS_r| \leq |\partial^i S_r|$, it is sufficient to show that $\partial^i RS_r \subseteq R\partial^i S_r$, since for all $S_r$, $|RS_r| = |S_r|$ (the replacement operator returns a set of the same size). This inclusion will be referred to as the *iterated crucial inclusion*. The following is a summary of the proof executed in *Larch* for the first inequality.

**1.** Prove $\partial^i RS_r \subseteq R\partial^i S_r$ by induction on $i$.
The basis subgoal is $\partial^0 RS_r \subseteq R\partial^0 S_r$ (for i=0), which reduces to $RS_r \subseteq RS_r$ by the rule $\partial^0 S_r = S_r$. Induction, of course, is on the natural numbers generated by 0 and the successor function.
At this point, the induction hypothesis may be assumed and temporarily added to the list of rules: $\partial^i RS_r \subseteq R\partial^i S_r$.
¿From here, the induction step must be proved: $\partial^{i+1} RS_r \subseteq R\partial^{i+1} S_r$.
*Larch* is unable to prove the induction step by normalization, and must be given guidance by the user.

**2.** To prove $\partial^{i+1} RS_r \subseteq \partial^{i+1} S_r$, it is sufficient to show $\partial^{i+1} RS_r \subseteq \partial R\partial^i S_r$ and $\partial R\partial^i S_r \subseteq R\partial^{i+1} S_r$. *Larch* has an easy time with the second inclusion, but the first one is another story. The crucial inclusion, $\partial RS_r \subseteq R\partial S_r$, must be instantiated with $\partial^i S_r$ in place of $S_r$ to obtain a term which normalizes to *true*.

**3.** After both inclusions are proved, the three sets are instantiated into the well-known transitive inclusion formula:

$$x \subseteq y \text{ and } y \subseteq z \Rightarrow x \subseteq z,$$

proving the inductive step.

**4.** It is an easy matter to obtain the first inequality here by taking critical pairs between the iterated crucial inclusion and the axioms on the replacement operator and on the sizes of sets for which inclusion holds.

The proof of the second inequality is similar, and is outlined in the program output in the Appendix.

# A Appendix: Proof of the Inequalities in *Larch*

```
set script-file cor
set script

% object classes necessary are natural numbers and sets
declare sort Set,Nat,NegNat

% here are declared prototype objects from their respective classes
declare variables x,y,z:Set, i,j,k:Nat, l:NegNat

%************************************************************
% Here are some standard operators for sets and natural numbers
%       \subseteq       for sets
%       \leq            inequality for nat numbers
%       0,s             generators for nat numbers
%************************************************************
declare operators
   __\subseteq__:  Set,Set -> Bool
   __\leq__:       Nat,Nat -> Bool
   0:                      -> Nat
   s:              Nat     -> Nat
  ..

% natural numbers are generated inductively by 0 and s
assert sort Nat generated by 0,s

%************************************************************
% These operators are specific to the proof
%       d               takes the boundary of a set
%       R               takes the replacement of a set
%       size            returns cardinality of a set
%************************************************************
declare operators
   d:              Nat,Set -> Set
   R:              Set     -> Set
   size:           Set     -> Nat
  ..

%************************************************************
% the first two axioms are standard set theory
% the next three axioms are related to the boundary
% operator:
% (3): the 0th boundary of a set is that set
% (4): the (i+1)th boundary of a set is the
```

```
%       boundary of the ith boundary of that set
% (5): if x is a subset of y, applying the boundary
%       to both sets preservers the subset relation
%**********************************************************
set name cor_axioms
assert
   x \subseteq x;
   x\subseteq y => size(x)\leq size(y);
   d(0,x)=x;
   d(s(i),x)=d(s(0),d(i,x));
   x\subseteq y => d(s(0),x)\subseteq d(s(0),y);
   ..

% axiom of transitivity of set inclusion
set name InclTrans
assert
   x\subseteq y \/ y \subseteq z => x \subseteq z;
   ..

%**********************************************************
% the Crucial Inclusion theorem assumed for the proof
% of the Corollary (given immunity because a subsequent fact
% has this theorem as a special case; we don't want to delete
% this theorem, since it is the main assumption)
%**********************************************************
set name CrucIncl
set immunity on
assert
   d(s(0),R(x)) \subseteq R(d(s(0),x));
   ..
set immunity off

%**********************************************************
% The ith boundary generalization of the Crucial Inclusion
%       The basis step of the induction is easy, but moving
% from the inductive hypothesis to the induction goal
% requires proving an instantiation of the Crucial Inclusion
% followed by the observation of transitivity on the
% subset relation.
%**********************************************************
set name CrucIncl_Iterated
prove d(i,R(x))\subseteq R(d(i,x)) by induction on i
    <> basis subgoal
    [] basis subgoal
    <> induction subgoal
```

```
    prove d(s(0),R(d(ic,x))) \subseteq R(d(s(ic),x))
      instantiate x by d(ic,x) in CrucIncl
      [] conjecture
    instantiate
       x by d(s(ic),R(x)),
       y by d(s(0),R(d(ic,x))),
       z by R(d(s(ic),x)) in InclTrans
      ..
    [] induction subgoal
  [] conjecture
qed

%**************************************************************
% the Corollary is easy now that the Iterated Crucial Inclusion
% is combined with the fact that set containment implies
% set cardinality boundedness
%**************************************************************
set name Corollary
prove size(d(i,R(x))) \leq size(R(d(i,x)))
  crit-pairs CrucIncl_Iterated  with cor_axioms
  [] conjecture
qed

%**************************************************************
%**************************************************************
%************** PART 2 OF COROLLARY *************************
%* The corollary has been proved for the boundary operator,
%* but it also must be proved for the coboundary operator.
%* This operator is denoted by inv_d, and
%* inv_d(i,x) is the same as d(-i,x)
%**************************************************************
set name cor_axioms_inv
declare operators
   inv_d:         Nat,Set -> Set
  ..

%**************************************************************
%* There are several more axioms needed for the
%* coboundary operator. The first two are given for
%* the boundary operator also.
%* The next two govern how the boundary and coboundary
%* operators interact.
%* Finally, the last two show that containment is preserved
%* on application of the replacement operator R, or the
%* coboundary operator.
```

```
%***********************************************************
assert
   inv_d(0,x) = x;
   inv_d(i,inv_d(s(0),x)) = inv_d(s(i),x);
   inv_d(s(0),d(s(0),x)) = x;
   d(s(0),inv_d(s(0),x)) \subseteq x;
   x\subseteq y => R(x) \subseteq R(y);
   x\subseteq y => inv_d(s(0),x) \subseteq inv_d(s(0),y);
  ..

%***********************************************************
%* The coboundary corollary will be proved by induction;
%* it is useful to have both the basis step and the i=1
%* step proved and available. Here is the i=1 step, named
%* the inverse crucial inclusion.
%***********************************************************
set name inv_CrucIncl
set immunity on
prove R(inv_d(s(0),x)) \subseteq inv_d(s(0),R(x))
  set immunity off
  prove d(s(0),R(inv_d(s(0),x))) \subseteq R(x)
    prove d(s(0),R(inv_d(s(0),x))) \subseteq R(d(s(0),inv_d(s(0),x)))
      [] conjecture
    prove R(d(s(0),inv_d(s(0),x))) \subseteq R(x)
      instantiate
         x by d(s(0),inv_d(s(0),x)),
         y by x in cor_axioms_inv.5
        ..
      [] conjecture
    instantiate
       x by d(s(0),R(inv_d(s(0),x))),
       y by R(d(s(0),inv_d(s(0),x))),
       z by R(x) in InclTrans
      ..
     [] conjecture
  instantiate
     x by d(s(0),R(inv_d(s(0),x))),
     y by R(x) in cor_axioms_inv.6
    ..
  [] conjecture
qed

%***********************************************************
%* Now that the i=1 step is done, it is time to do the
%* induction for the general case.  The induction step
```

```
%* is proved by applying the i=1 step and transitivity
%* of set inclusion for three useful sets
%***************************************************************
set immunity off
set name inv_CrucIncl_Iterated
prove R(inv_d(i,x)) \subseteq inv_d(i,R(x)) by induction on i
    <> basis subgoal
    [] basis subgoal
    <> induction subgoal
    assert inv_d(s(0),inv_d(ic,x))=inv_d(s(ic),x)
    prove R(inv_d(s(ic),x)) \subseteq inv_d(s(0),R(inv_d(ic,x)))
      set immunity on
      instantiate x by inv_d(ic,x) in inv_CrucIncl.1
      set immunity off
      critical-pairs inv_CrucIncl* with inv_CrucIncl*
      [] conjecture
    prove inv_d(s(0),R(inv_d(ic,x))) \subseteq inv_d(s(ic),R(x))
      instantiate
         x by R(inv_d(ic,x)),
         y by inv_d(ic,R(x)) in cor_axioms_inv.6
        ..
      [] conjecture
    instantiate
       x by R(inv_d(s(ic),x)),
       y by inv_d(s(0),R(inv_d(ic,x))),
       z by inv_d(s(ic),R(x)) in InclTrans
      ..
    [] induction subgoal
  [] conjecture
qed

%***************************************************************
%* Once the iterated inverse Crucial Inclusion is obtained,
%* It is an easy matter to show the following corollary,
%* which obtains a set cardinality bound by using the
%* inclusion above.
%**************************************************************
set name inv_Corollary
prove size(R(inv_d(i,x))) \leq size(inv_d(i,R(x)))
  instantiate
     x by R(inv_d(i,x)),
     y by inv_d(i,R(x)) in cor_axioms.2
    ..
  [] conjecture
qed
```

# References


[1] Ellis, R., “A Kruskal-Katona Theorem for Cubical Complexes”, preprint (1996).

[2] Garland, S.J. , and Guttag, J.V., “A Guide to LP, The Larch Prover”, Massachusetts Institute of Technology (1993).

[3] Lindström, B., “The optimal number of faces in cubical complexes”, *Arkiv för Matematik* bf 9 #24 (1972), 245-257.