Verifying Burrows Wheeler Transformation In Coq

Hanlin He, Sourav Dasgupta, Paul Perrot, Alan Padilla Chua

December 14, 2017

1 Motivation

The idea that spawned this research project was given by Dr. Kevin Hamlen. When verifying security implementations, compression algorithms seemed to be overlooked. This left a security gap that is to be trusted without any formal verification. Our initial goal was to verify a full compression algorithm from top to bottom. However, we narrowed our search to *bzip2*'s compression stack and saw that its core algorithm depended on the Burrows Wheeler Transformation[].

BWT (Burrows Wheeler Transformation), has some important security properties related to integrity. It holds that BWT is a reversible transformation that has no side effects on the data. We saw BWT to be of great potential to be verified in Coq. Its full verification could narrow trusted code base of *bzip2*'s compression and allow future teams to tackle the full verification of *bzip2*.

2 Preliminaries

Our Coq implementation has been tested on the CoqIDE version 8.6.1 for Windows, Mac, and Linux.

3 Project Summary

3.1 Approach

For our implementation of BWT in Coq we decided to prove that the transformation is lossless; Meaning that for any string we apply the forward BWT on, we can subsequently perform the inverse BWT on that result and get the original string back. In order to accomplish this and be able to reason about it, we implemented the forward BWT and inverse BWT in Coq. Before getting to these two implementations we need to discuss some more general, but still important, parts of our implementation.

Dr. Kevin Hamlen mentioned that proving theorems about lists could lead to complex issues. So instead of using lists, for our main implementation we chose to implement strings as mappings ($str : nat \rightarrow option\ nat$) and matrices as a mapping of mappings ($matrix : nat \rightarrow str$). It is worth noting that we did create a generic implementation of BWT using lists but did not attempt to prove any lemmas about it. Our proofs are solely related to the string mapping version of our Coq code.

3.1.1 Forward BWT

Given a string as input, we first convert a string into a str mapping, and construct the matrix by right shift the str mapping, then assuming a given sort function that sort the matrix lexicographically, and return the last column of the sorted matrix as the result string.

3.1.2 Inverse BWT

Given a string and an index as input, we assume a given sort function that sort the string lexicographically, we first generate the index mapping pi between the original string and the sorted string, and then recursively generates a sequence of indices using pi and the given index, each of which corresponds to one character in the original string, and apply all the index with the given string to regenerate the original before BWT.