Verified Deflate. What's Next?

Christoph-Simon Senjak

Lehr- und Forschungseinheit für Theoretische Informatik
Institut für Informatik
Ludwig-Maximilians-Universität München
Oettingenstr.67, 80538 München

PUMA Workshop 2017

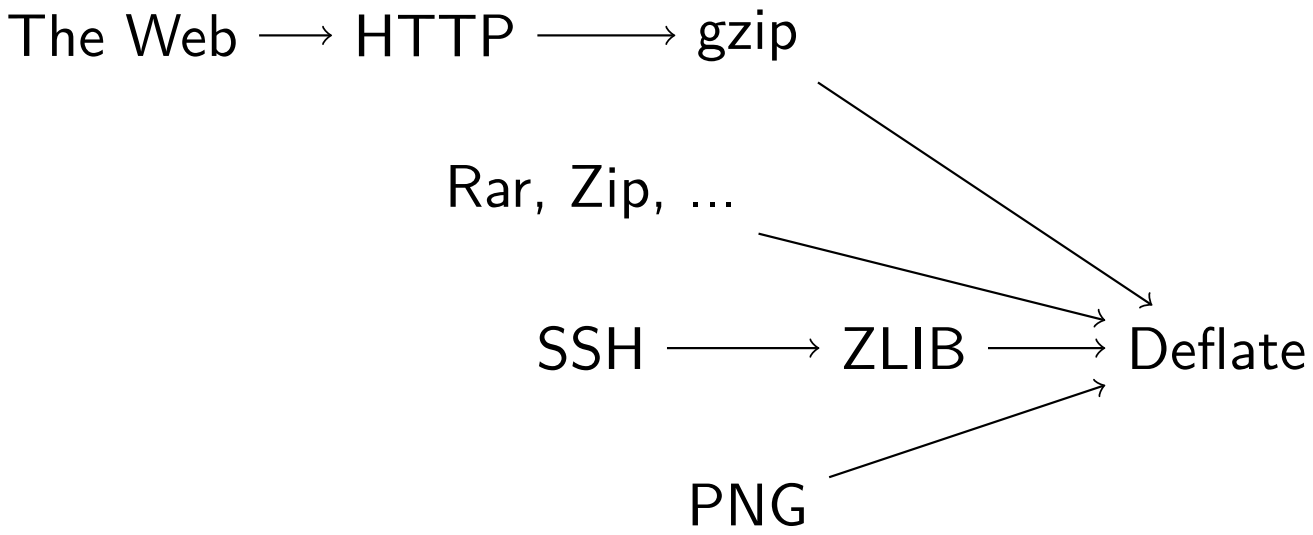
Foreword

- We formalized the Deflate compression standard in the Coq proof assistant.
- We tried to utilize program extraction to create a fully verified implementation of a compression and decompression algorithm.
- However, there are several shortcomings. In this talk, we will focus on this future work.

Deflate – Basics

- Specified in RFC 1951.
- RFC 1952 specifies the GZIP file format, and RFC 1950 specifies the ZLIB file format. Both are often confused with Deflate.
- Can make use of Huffman-Codings and Run-length encoding.
- Does not require any specific compression algorithm, even though some are recommended.

Why Deflate?



Main implementation is the zlib (zlib.net).
Though the first release was 1995, in 2005 there were still security vulnerabilities.
There are still lots of bugfixes with every version.

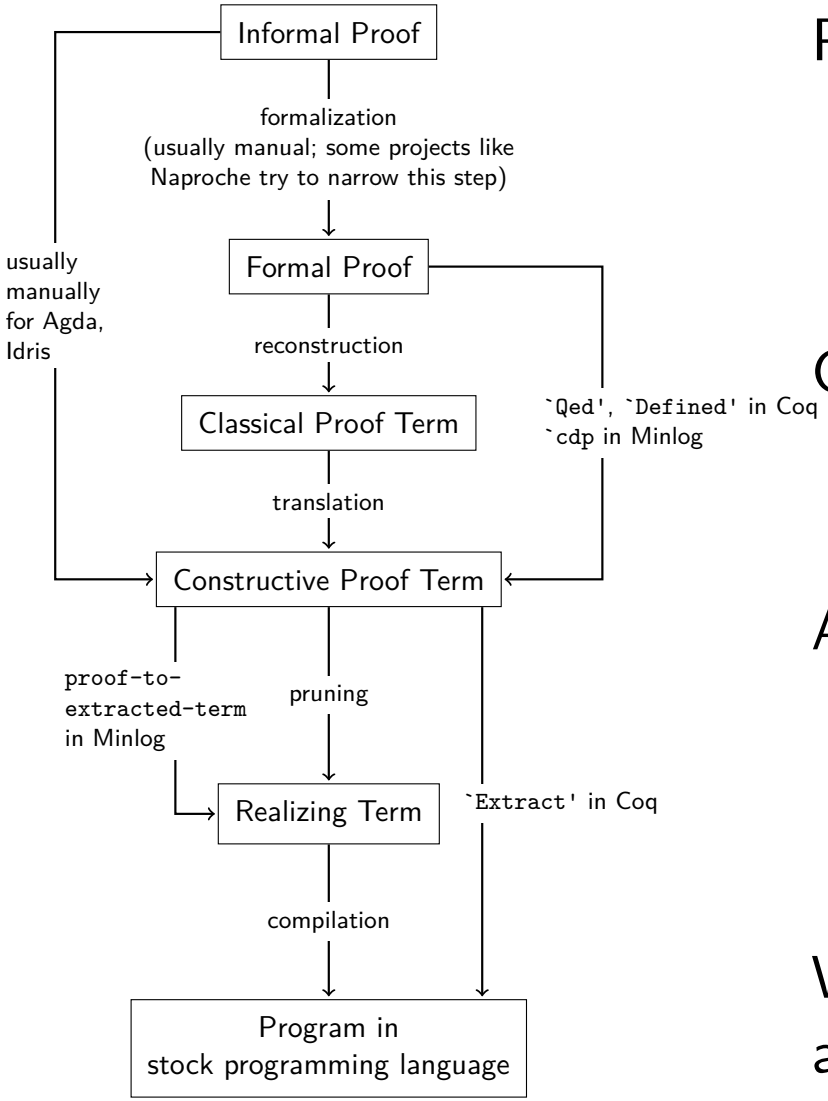
Deflate - A very brief overview

An informal illustration of the format:

```
Deflate              ::=  ('0' Block)* '1' Block (0|1)*
Block                ::=  '00' UncompressedBlock |
                          '01' DynamicallyCompBl |
                          '10' StaticallyCompBl
UncompressedBlock    ::=  length ~length bytes
StaticallyCompBl     ::=  CompBl(standard coding)
DynamicallyCompBl    ::=  header coding CompBl(coding)
CompBl(c)            ::=  [^256]* 256
```

Does not require any specific compression algorithm, even though some are recommended $\Rightarrow$ we use encoding relations rather than directly writing a function. We then prove their realizability and extract algorithms.

Program Extraction



Pros of extraction:

- Sophisticated formats should come with (at least informal) correctness proofs anyway.

Cons of extraction:

- Proofs must be (mostly) constructive.

Alternatives in Coq:

- Writing the functions dependently typed.
- Verifying functions a posteriori.

We sometimes mix these styles when apropriate.

What we have

- Mathematical specification of a compression format that is very likely Deflate
  - Tested it with lots of randomized data.
  - And with real-life data.
  - However, the specification is axiomatic, and the standard is informally specified.
- Implementation of a compression and a decompression algorithm through program extraction
  - Verified to be inverse to each other.
  - Still slow, but “awaitable”, lots of potential for optimization.

However, in this talk, we want to focus on what hasn't been done, and how this can be achieved in the future.

ZLib Drop-In Replacement

- We can compress and decompress by commandline.
- We can also embed it inside Haskell programs.
- However, we do not support the ZLib-API, so we cannot use it as a drop-in-replacement.
- It is possible to embed Haskell into C. Maybe it would also be possible to extract to some other language (OCaml is a canonical candidate).

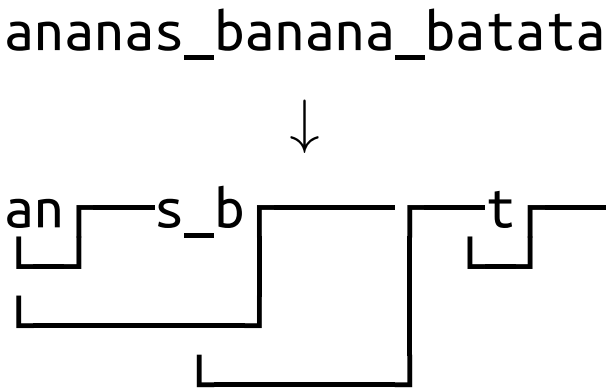
Laziness

- The algorithms rely on lazy evaluation. This simplified the work.
- For the largest part, it should be replacable by strict algorithms. Except for lazy I/O.
- Alternatives for lazy I/O would be formalizations of *Iteratees*, *Pipes* or *Machines*.

The Specification is Complicated

- The original specification is informal, and not always entirely clear. We had to test some parts before we knew how they worked.
- The topic is complex, and therefore, things cannot be made arbitrarily easy.
- However, it may be possible to formalize some parts of the specification in a more simple way.

Backreference resolution



- This is the slowest part of the implementation.
- We have
  - a purely functional and efficient resolver
  - an implementation using diffarrays

  but none of them are verified yet.
- We currently use an implementation that utilizes a list structure with recursive slowdown. It is the main bottleneck right now.

Conclusion

- To our best knowledge, this is the first realistic implementation of a verified compression and decompression algorithm.
- The implementation has weaknesses, but they can be overcome.
- We hope that it will be developed further.