Using DiffArrays to optimize Decompression

Christoph-Simon Senjak

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

ABM 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, making it efficient in the absence of destructive operations is hard.
- DiffArrays to the rescue!

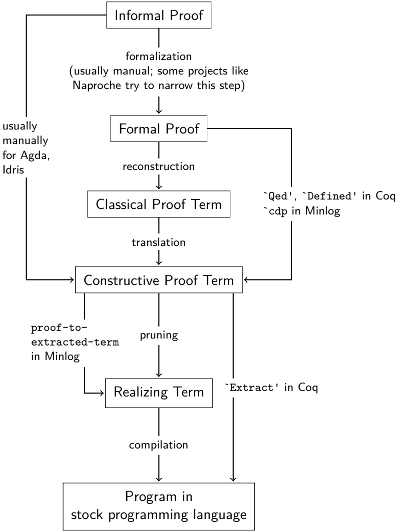
Deflate - A very brief overview

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
```

Specified in RFC 1951. Allows for Lempel-Ziv-compression (backreferences) and Huffman coding. Widely used (GZip, Zip, HTTP, SSH, etc).

Program Extraction



Program Extraction

Pros and cons of program extraction:

- Sophisticated formats should come with (at least informal) correctness proofs anyway. However, reality looks different.
- Proofs must be (mostly) constructive.

Alternatives in Coq:

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

We sometimes mix these styles when apropriate.

Backreferences

- A backreference is a pair $\langle l, d \rangle$ of a length l (number of bytes to be copied) and a distance d (number of recently extracted bytes that have to be skipped).
- Example:

```
ananas_banana_batata
```

Backreferences

- A backreference is a pair $\langle l, d \rangle$ of a length l (number of bytes to be copied) and a distance d (number of recently extracted bytes that have to be skipped).
- Example:

```
  2
ananas_banana_batata
|  |
+--+
```

Backreferences

- A backreference is a pair $\langle l, d \rangle$ of a length l (number of bytes to be copied) and a distance d (number of recently extracted bytes that have to be skipped).
- Example:

```
  22
ananas_banana_batata
 |  |
 +--+
```

Backreferences

- A backreference is a pair $\langle l, d \rangle$ of a length l (number of bytes to be copied) and a distance d (number of recently extracted bytes that have to be skipped).
- Example:

```
  222
ananas_banana_batata
  |  |
  +--+
```

Backreferences

- A backreference is a pair $\langle l, d \rangle$ of a length l (number of bytes to be copied) and a distance d (number of recently extracted bytes that have to be skipped).
- Example:

```
  222   8
ananas_banana_batata
|        |
+--------+
```

Backreferences

- A backreference is a pair $\langle l, d \rangle$ of a length l (number of bytes to be copied) and a distance d (number of recently extracted bytes that have to be skipped).
- Example:

```
  222   88
ananas_banana_batata
 |        |
 +--------+
```

Backreferences

- A backreference is a pair $\langle l, d \rangle$ of a length l (number of bytes to be copied) and a distance d (number of recently extracted bytes that have to be skipped).
- Example:

```
  222   888
ananas_banana_batata
  |       |
  +-------+
```

Backreferences

- A backreference is a pair $\langle l, d \rangle$ of a length l (number of bytes to be copied) and a distance d (number of recently extracted bytes that have to be skipped).
- Example:

```
  222   8888
ananas_banana_batata
   |         |
   +--------+
```

Backreferences

- A backreference is a pair $\langle l, d \rangle$ of a length l (number of bytes to be copied) and a distance d (number of recently extracted bytes that have to be skipped).
- Example:

```
  222   88888
ananas_banana_batata
    |        |
    +--------+
```

Backreferences

- A backreference is a pair $\langle l, d \rangle$ of a length l (number of bytes to be copied) and a distance d (number of recently extracted bytes that have to be skipped).
- Example:

```
  222   888887
ananas_banana_batata
      |       |
      +-------+
```

Backreferences

- A backreference is a pair $\langle l, d \rangle$ of a length l (number of bytes to be copied) and a distance d (number of recently extracted bytes that have to be skipped).
- Example:

```
  222   8888877
ananas_banana_batata
       |      |
       +------+
```

Backreferences

- A backreference is a pair $\langle l, d \rangle$ of a length l (number of bytes to be copied) and a distance d (number of recently extracted bytes that have to be skipped).
- Example:

```
  222   88888777
ananas_banana_batata
        |       |
        +-------+
```

Backreferences

- A backreference is a pair $\langle l, d\rangle$ of a length l (number of bytes to be copied) and a distance d (number of recently extracted bytes that have to be skipped).
- Example:

```
  222   88888777 2
ananas_banana_batata
               |  |
               +-+
```

Backreferences

- A backreference is a pair $\langle l, d \rangle$ of a length l (number of bytes to be copied) and a distance d (number of recently extracted bytes that have to be skipped).
- Example:

```
  222   88888777 22
ananas_banana_batata
               |  |
               +-+
```

Backreferences

- A backreference is a pair $\langle l, d \rangle$ of a length l (number of bytes to be copied) and a distance d (number of recently extracted bytes that have to be skipped).
- Example:

```
  222   88888777 222
ananas_banana_batata
                |  |
                +--+
```

Backreferences

- A backreference is a pair $\langle l, d \rangle$ of a length l (number of bytes to be copied) and a distance d (number of recently extracted bytes that have to be skipped).
- Example:

```
   3          5     3     3
  222     88888777  222
ananas_banana_batata
```

Backreferences

- A backreference is a pair $\langle l, d \rangle$ of a length l (number of bytes to be copied) and a distance d (number of recently extracted bytes that have to be skipped).
- Example:

```
   3        5    3    3
  222     88888777  222
ananas_banana_batata
```

$\Rightarrow$ `an` $\langle 3, 2 \rangle$ `s_b` $\langle 5, 8 \rangle$ $\langle 3, 7 \rangle$ `t` $\langle 3, 2 \rangle$

Backreferences

- A backreference is a pair $\langle l, d \rangle$ of a length l (number of bytes to be copied) and a distance d (number of recently extracted bytes that have to be skipped).
- Example:

```
   3      5   3   3
  222   88888777 222
ananas_banana_batata
```

$\Rightarrow$ `an` $\langle 3, 2 \rangle$ `s_b` $\langle 5, 8 \rangle$ $\langle 3, 7 \rangle$ `t` $\langle 3, 2 \rangle$

- This is the slowest part of the decompression algorithm.
- We have
  - a purely functional and fast resolver, which is not verified
  - an implementation that utilizes a list structure with recursive slowdown, which is slow
  - an implementation using diffarrays, which is fast

Resolution in General

- Use some map structure as buffer to save the last *n* decompressed bytes.
- If a backreference points to it, copy it to the front.
- For example, a naïve way of doing it, using a list as buffer (the second argument), is:

```
resolve :: [Either a (Int, Int)] -> [a] -> [a]
resolve [] _ = []
resolve ((Left b) : r) x = b : resolve r (b : x)
resolve (Right (0, _) : r) x = resolve r x
resolve (Right (l, d) : r) x =
    let b = x !! (d - 1) -- get (d-1)th element from x
    in b : resolve (Right (l-1, d) : r) (b : x)
```

Queue of Doom

- In an imperative setting, we would use a ring buffer. However, in the functional setting, even with DiffArrays, this makes things more compliated.
- We only need to save a 32KiB history, as backreferences are limited. We therefore can put two objects of 32KiB size in a **Queue of Doom**
- Example of a queue of doom with 3 elements (instead of 32 KiB): `start` `[] []`

Queue of Doom

- In an imperative setting, we would use a ring buffer. However, in the functional setting, even with DiffArrays, this makes things more compliated.
- We only need to save a 32KiB history, as backreferences are limited. We therefore can put two objects of 32KiB size in a **Queue of Doom**
- Example of a queue of doom with 3 elements (instead of 32 KiB): `push 1` `[1] []`

Queue of Doom

- In an imperative setting, we would use a ring buffer. However, in the functional setting, even with DiffArrays, this makes things more compliated.
- We only need to save a 32KiB history, as backreferences are limited. We therefore can put two objects of 32KiB size in a **Queue of Doom**
- Example of a queue of doom with 3 elements (instead of 32 KiB): `push 2` `[2;1] []`

Queue of Doom

- In an imperative setting, we would use a ring buffer. However, in the functional setting, even with DiffArrays, this makes things more compliated.
- We only need to save a 32KiB history, as backreferences are limited. We therefore can put two objects of 32KiB size in a **Queue of Doom**
- Example of a queue of doom with 3 elements (instead of 32 KiB): `push 3` `[3;2;1] []`

Queue of Doom

- In an imperative setting, we would use a ring buffer. However, in the functional setting, even with DiffArrays, this makes things more compliated.
- We only need to save a 32KiB history, as backreferences are limited. We therefore can put two objects of 32KiB size in a **Queue of Doom**
- Example of a queue of doom with 3 elements (instead of 32 KiB): `push 4` `[4]` `[3;2;1]` `[]` $\rightarrow$ ☠

Queue of Doom

- In an imperative setting, we would use a ring buffer. However, in the functional setting, even with DiffArrays, this makes things more compliated.
- We only need to save a 32KiB history, as backreferences are limited. We therefore can put two objects of 32KiB size in a **Queue of Doom**
- Example of a queue of doom with 3 elements (instead of 32 KiB): `push 5` `[5;4] [3;2;1]`

Queue of Doom

- In an imperative setting, we would use a ring buffer. However, in the functional setting, even with DiffArrays, this makes things more compliated.
- We only need to save a 32KiB history, as backreferences are limited. We therefore can put two objects of 32KiB size in a **Queue of Doom**
- Example of a queue of doom with 3 elements (instead of 32 KiB): `push 6` `[6;5;4] [3;2;1]`

Queue of Doom

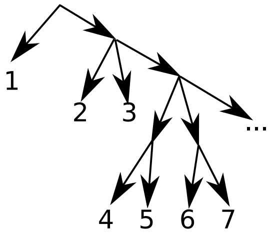
- In an imperative setting, we would use a ring buffer. However, in the functional setting, even with DiffArrays, this makes things more compliated.
- We only need to save a 32KiB history, as backreferences are limited. We therefore can put two objects of 32KiB size in a **Queue of Doom**
- Example of a queue of doom with 3 elements (instead of 32 KiB): `push 7` `[7]` `[6;5;4]` `[3;2;1]` $\rightarrow$ ☠

Using Recursive Slowdown

- "ExpList":

```
Inductive ExpList (A : Set) : Set :=
| Enil : ExpList A
| Econs1 : A -> ExpList (A * A) -> ExpList A
| Econs2 : A -> A -> ExpList (A * A) -> ExpList
```

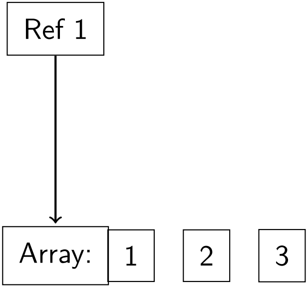
- The advantage of `ExpLists` is that `nth` and `cons` consume logarithmic time.



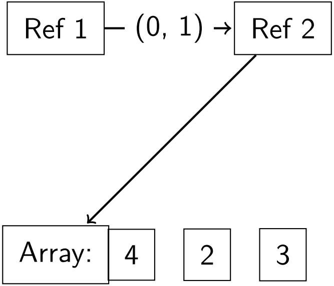
DiffArrays

- The advantage of imperative arrays with destructive updates is their $O(1)$ modification and read.
- It is possible to embed destructive operations into linear type systems. However, Coq is not linear, and neither are Haskell (which uses Monads instead) and OCaml (which is impure).
- However, we may still write the code in a linear fashion, so it is theoretically possible for the compiler to optimize the code.
- DiffArrays are fast exactly when they are used in a linear fashion, and slow otherwise.

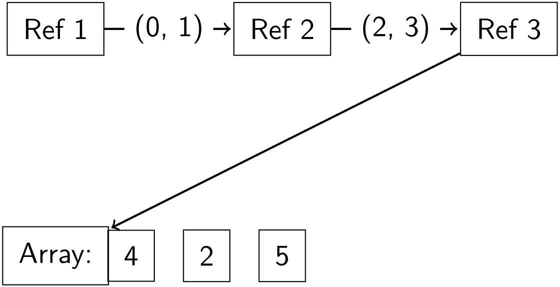
DiffArrays



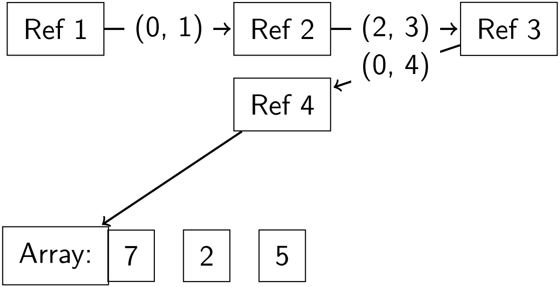
DiffArrays



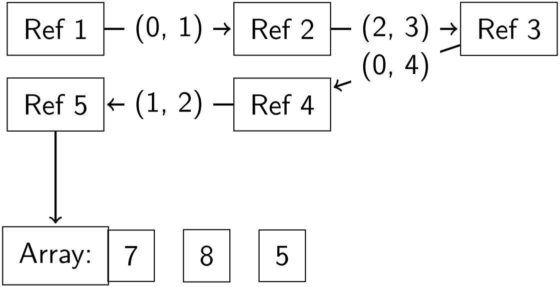
DiffArrays



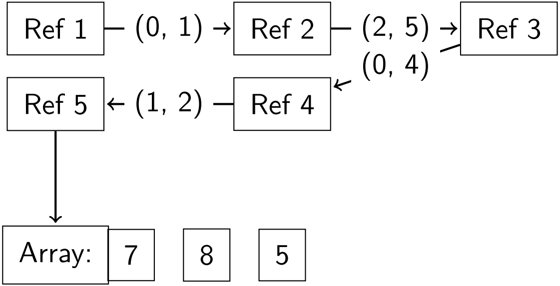
DiffArrays



DiffArrays



DiffArrays



⇒ The old versions can be reconstructed in $O(t)$, where t is the number of versions.

⇒ Read- and Write-Access to the newest reference takes $O(1)$, as for imperative arrays.

DiffStacks

On top of the diffarrays, we put a stack structure in the usual way. Our formalization in Coq then has axioms like

```
Axiom DSPush : forall (A : Set) (a : A)
                (ds : DiffStack A),
                     DiffStack A.

Axiom DSNth : forall (A : Set) (n : nat)
                     (ds : DiffStack A)
                     (default : A),
    A * DiffStack A.

Axiom DSNthFakeLinear :
    forall {A : Set} n ds d,
      snd (@DSNth A n ds d) = ds.
```

The last axiom says that the DiffStack after reading one element from it is the same as before.

Benchmarks

The Canterbury Corpus is a standardized set of files for benchmarking compression and decompression algorithms. Decompressing the files compressed with gzip results in the following table:

Time (s) ExpList	Time (s) DiffArray	File
0.27	0.11	grammar.lsp
4.22	0.25	fields.c
86.73	0.56	cp.html
0.42	0.14	xargs.1
177.54	0.85	sum
628.5	3.43	asyoulik.txt
727.25	4.05	alice29.txt
1958.08	10.72	lcet10.txt
2177.95	12.78	plrabn12.txt
2241.7	7.5	ptt5
3540.27	17.42	kennedy.xls

Conclusion, Related Work

- DiffArrays helped to boost efficiency without really losing any formal guarantees, by only slightly increasing the trusted codebase.
- This seems like a good way to embed some linear type constructs into purely functional code without linear type system.
- To lift this to a formal level, there has been done some work in "Adjustable References" by Vafeiadis, V.