Formally Verified Quite OK Image Format

With Stainless

Mario Bucev Viktor Kunčak

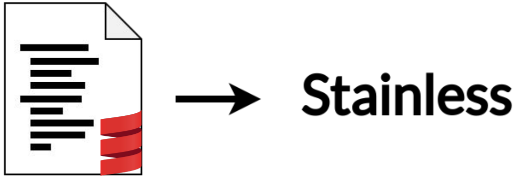
EPFL IC LARA

FMCAD'22, October 21st 2022

The Quite OK Image Format (QOI)

- Invented by Dominic Szablewski, announced a first version the 24th Nov. 2021
  - Finalized the 20th December
- Efficient and simple lossless image compression algorithm
  - C implementation with 311 LOC
  - Similar compression ratio as `libpng`
  - 3-4x and 30x higher throughput for decoding and encoding
- Only 4 methods to encode pixels!
  - RLE, dictionary, Δ color, full RGB(A)

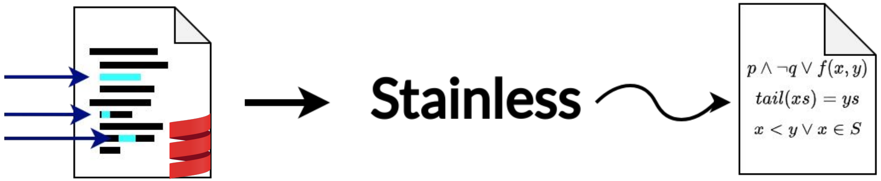
Stainless



→ Stainless

Scala program
+ contracts

Stainless

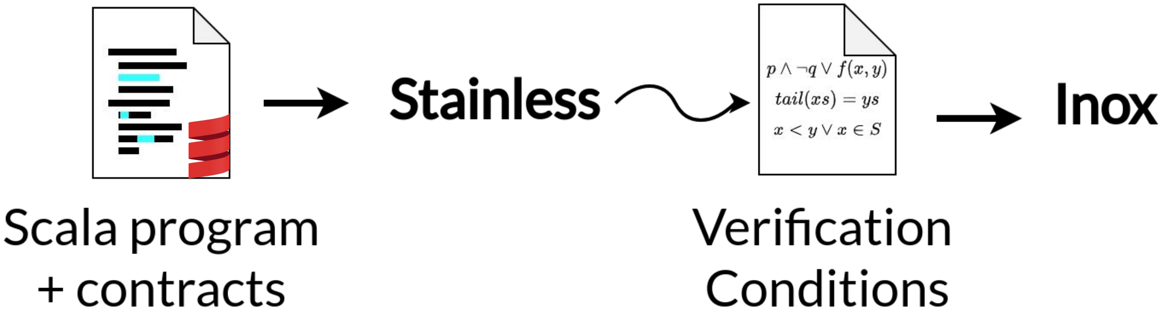


Scala program
+ contracts

Stainless

Verification
Conditions

Stainless



Stainless

Inox

$p \wedge \neg q \vee f(x, y)$

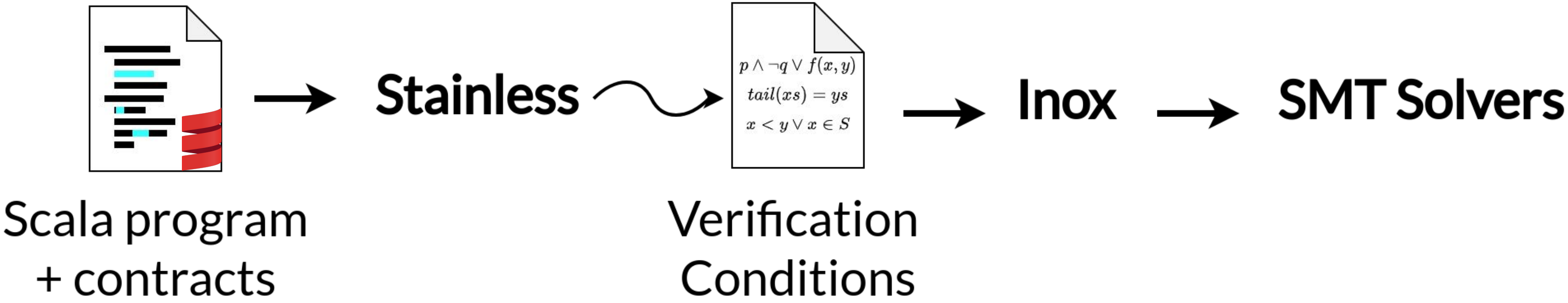
$tail(xs) = ys$

$x < y \vee x \in S$

Scala program
+ contracts

Verification
Conditions

Stainless



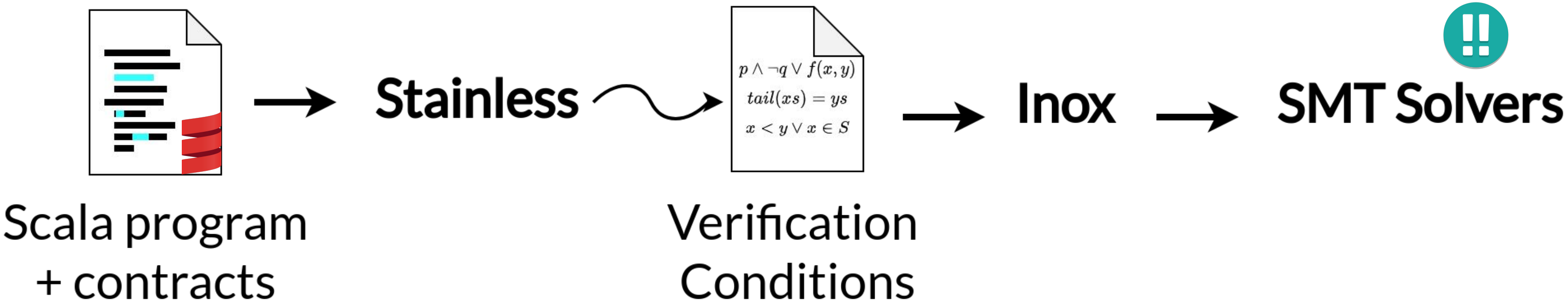
Scala program + contracts → Stainless ⤳ Verification Conditions → Inox → SMT Solvers

$p \land \neg q \lor f(x,y)$

$tail(xs) = ys$

$x < y \lor x \in S$

Stainless



Scala program + contracts

Stainless

$p \land \neg q \lor f(x, y)$

$tail(xs) = ys$

$x < y \lor x \in S$

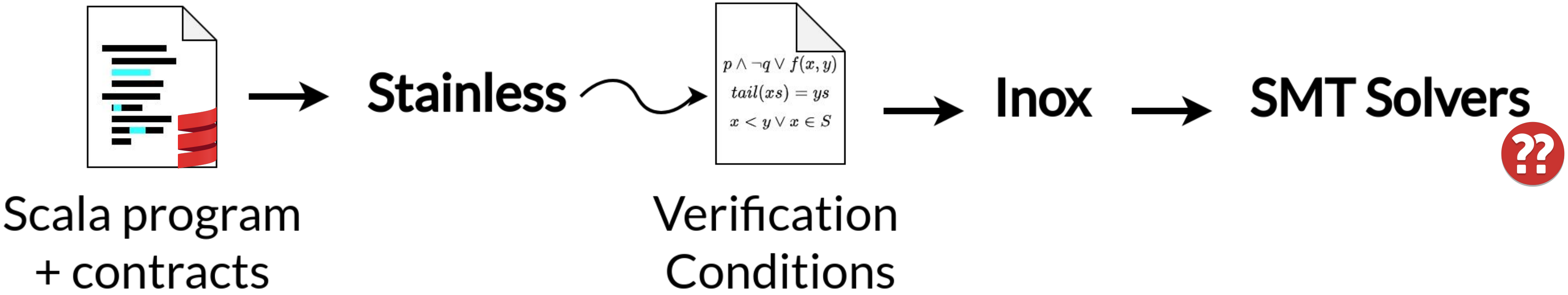
Verification Conditions

Inox

SMT Solvers

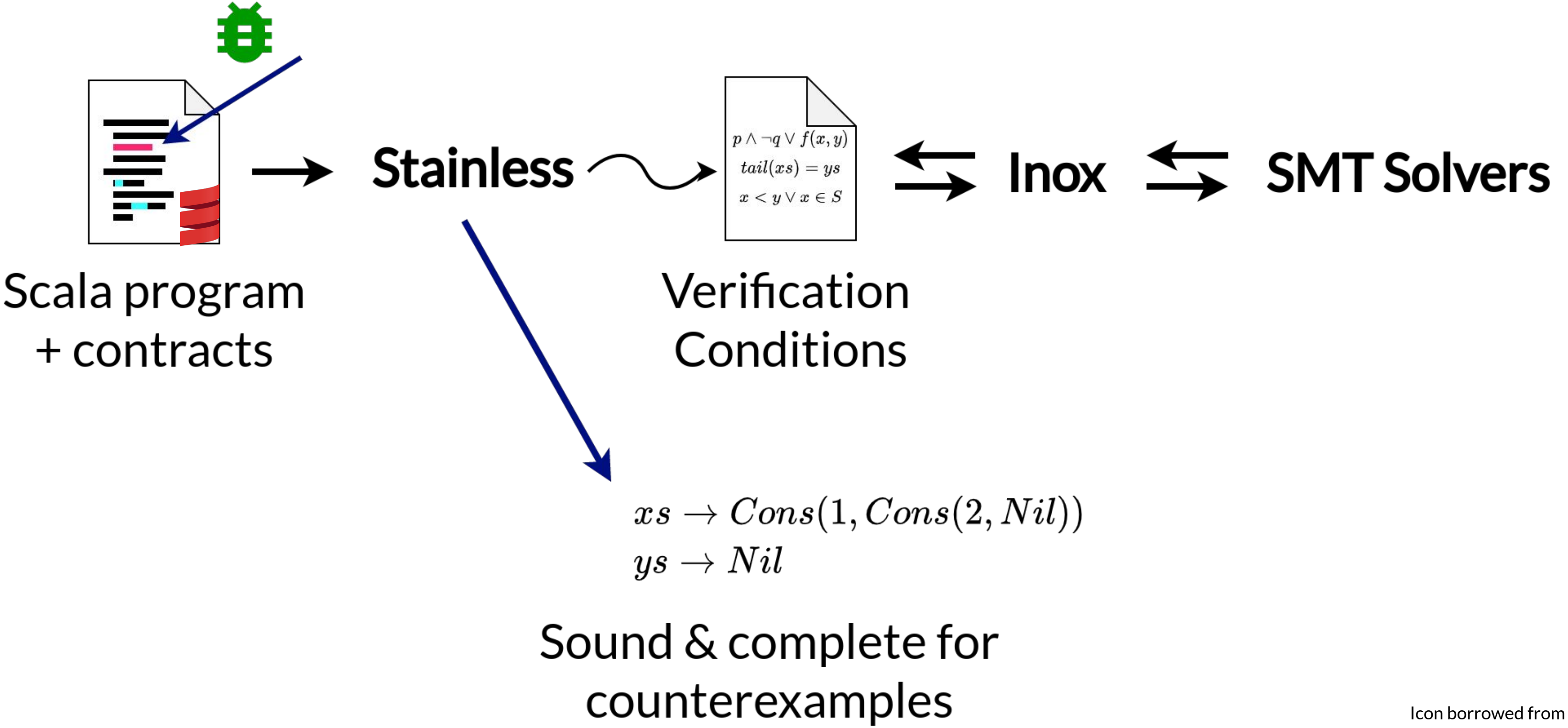
Icon borrowed from chess.com

Stainless



Icon borrowed from chess.com

Stainless



Scala program + contracts

Stainless

$p \wedge \neg q \vee f(x, y)$

$tail(xs) = ys$

$x < y \vee x \in S$

Verification Conditions

Inox

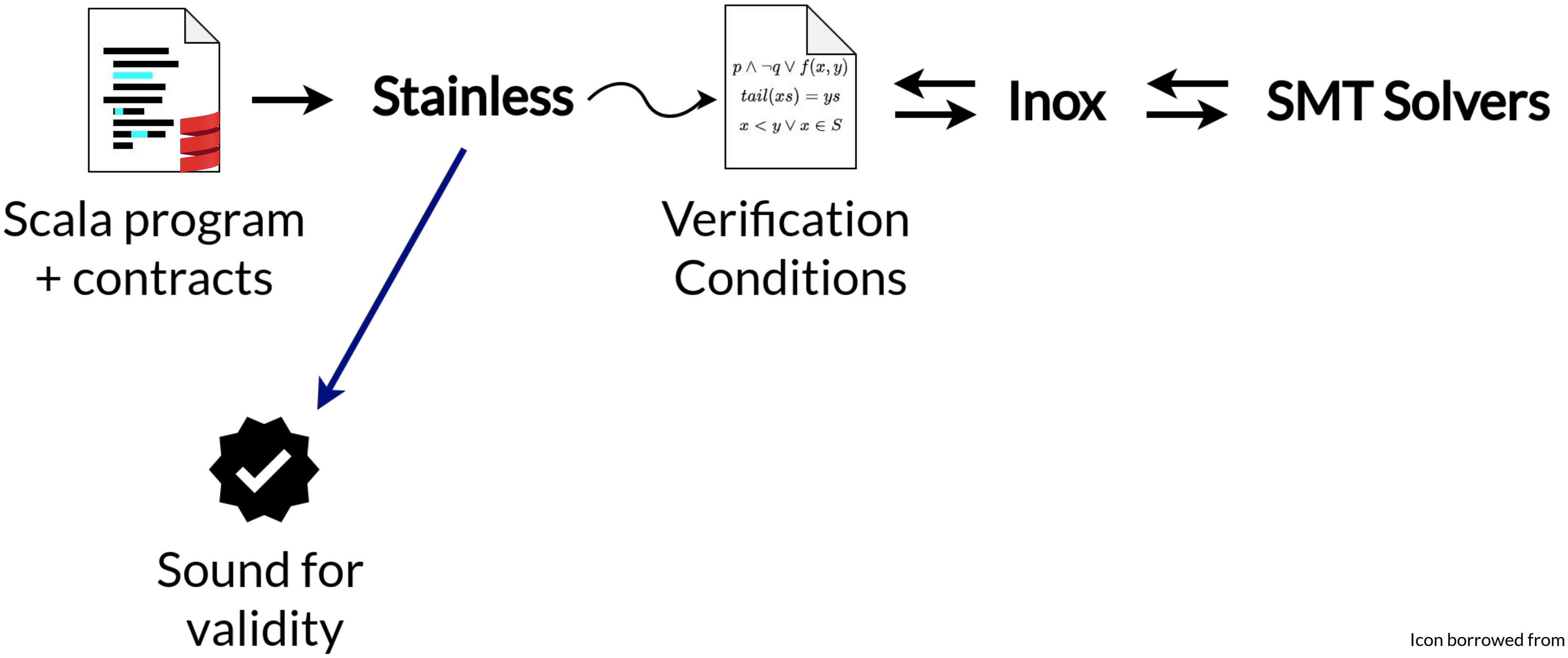
SMT Solvers

$$xs \rightarrow Cons(1, Cons(2, Nil))$$
$$ys \rightarrow Nil$$

Sound & complete for counterexamples

Icon borrowed from Google Fonts

Stainless



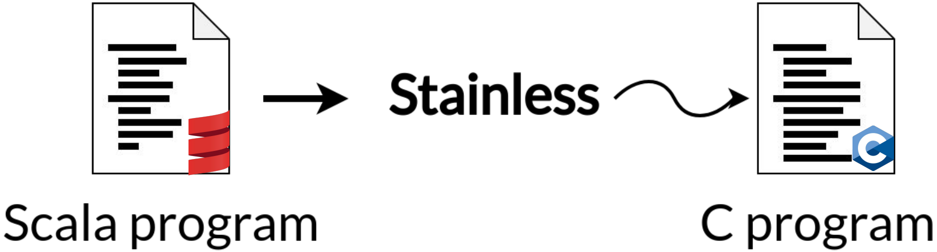
Scala program
+ contracts

Verification
Conditions

Sound for
validity

Icon borrowed from Google Fonts

Stainless



Scala program

Stainless

C program

Stainless & QOI: what to verify

- **Algorithmic correctness:** decoding is the inverse of encoding
  - Why is invertibility the right high-level property to check?
  - Because it guarantees no data loss
  - For compression, it can be empirically checked
- **Enforced properties:** runtime safety, termination, invariants

Stainless & QOI

```
def decodeEncodeIsIdentityThm(pixels: Array[Byte], w: Long, h: Long, chan: Long): Boolean = {
  require(inputInv(pixels, w, h, chan))
  val EncodedResult(bytes, outPos) = encode(pixels, w, h, chan)

  decode(bytes, outPos) match
    case SomeMut(DecodedResult(decodedPixels, ww, hh, cchan)) =>
      ww == w &&
      hh == h &&
      cchan == chan &&
      arraysEq(pixels, decodedPixels, 0, pixels.length)
    case NoneMut() => false
}.holds
```

Stainless & QOI

```
def decodeEncodeIsIdentityThm(pixels: Array[Byte], w: Long, h: Long, chan: Long) : Boolean = {
  require(inputInv(pixels, w, h, chan))
  val EncodedResult(bytes, outPos) = encode(pixels, w, h, chan)

  decode(bytes, outPos) match
    case SomeMut(DecodedResult(decodedPixels, ww, hh, cchan)) =>
      ww == w &&
      hh == h &&
      cchan == chan &&
      arraysEq(pixels, decodedPixels, 0, pixels.length)
    case NoneMut() => false
}.holds
```

∀`pixels,w,h,chan`
within bounds

Stainless & QOI

```
def decodeEncodeIsIdentityThm(pixels: Array[Byte], w: Long, h: Long, chan: Long): Boolean = {
  require(inputInv(pixels, w, h, chan))
  val EncodedResult(bytes, outPos) = encode(pixels, w, h, chan)

  decode(bytes, outPos) match
    case SomeMut(DecodedResult(decodedPixels, ww, hh, cchan)) =>
      ww == w &&
      hh == h &&
      cchan == chan &&
      arraysEq(pixels, decodedPixels, 0, pixels.length)
    case NoneMut() => false
}.holds
```

this expression is `true`

Stainless & QOI

```
def decodeEncodeIsIdentityThm(pixels: Array[Byte], w: Long, h: Long, chan: Long): Boolean = {
  require(inputInv(pixels, w, h, chan))
  val EncodedResult(bytes, outPos) = encode(pixels, w, h, chan)

  decode(bytes, outPos) match
    case SomeMut(DecodedResult(decodedPixels, ww, hh, cchan)) =>
      ww == w &&
      hh == h &&
      cchan == chan &&
      arraysEq(pixels, decodedPixels, 0, pixels.length)
    case NoneMut() => false
}.holds
```

Decoding what we just encoded must…

Stainless & QOI

```scala
def decodeEncodeIsIdentityThm(pixels: Array[Byte], w: Long, h: Long, chan: Long): Boolean = {
  require(inputInv(pixels, w, h, chan))
  val EncodedResult(bytes, outPos) = encode(pixels, w, h, chan)

  decode(bytes, outPos) match
    case SomeMut(DecodedResult(decodedPixels, ww, hh, cchan)) =>
      ww == w &&
      hh == h &&
      cchan == chan &&
      arraysEq(pixels, decodedPixels, 0, pixels.length)
    case NoneMut() => false
}.holds
```

always succeed (decoding cannot fail)

Stainless & QOI

```
def decodeEncodeIsIdentityThm(pixels: Array[Byte], w: Long, h: Long, chan: Long): Boolean = {
  require(inputInv(pixels, w, h, chan))
  val EncodedResult(bytes, outPos) = encode(pixels, w, h, chan)

  decode(bytes, outPos) match
    case SomeMut(DecodedResult(decodedPixels, ww, hh, cchan)) =>
      ww == w &&
      hh == h &&
      cchan == chan &&
      arraysEq(pixels, decodedPixels, 0, pixels.length)
    case NoneMut() => false
}.holds
```

yield an image whose dimensions match the original one…

Stainless & QOI

```
def decodeEncodeIsIdentityThm(pixels: Array[Byte], w: Long, h: Long, chan: Long): Boolean = {
  require(inputInv(pixels, w, h, chan))
  val EncodedResult(bytes, outPos) = encode(pixels, w, h, chan)

  decode(bytes, outPos) match
    case SomeMut(DecodedResult(decodedPixels, ww, hh, cchan)) =>
      ww == w &&
      hh == h &&
      cchan == chan &&
      arraysEq(pixels, decodedPixels, 0, pixels.length)
    case NoneMut() => false
}.holds
```

…with the same # of channels…

Stainless & QOI

```
def decodeEncodeIsIdentityThm(pixels: Array[Byte], w: Long, h: Long, chan: Long): Boolean = {
  require(inputInv(pixels, w, h, chan))
  val EncodedResult(bytes, outPos) = encode(pixels, w, h, chan)

  decode(bytes, outPos) match
    case SomeMut(DecodedResult(decodedPixels, ww, hh, cchan)) =>
      ww == w &&
      hh == h &&
      cchan == chan &&
      arraysEq(pixels, decodedPixels, 0, pixels.length)
    case NoneMut() => false
}.holds
```

...and identical pixels

Verification endeavor

- ~4 to 5 weeks to implement & formally verify
- A first version using imperative loops was quickly out
  - Proving runtime safety was easy
  - Specifying interesting properties was inconvenient :(
- Multiple rewrites were needed to achieve invertibility
  - Leverage recursion instead and split code parts into small functions
- Verification cache was helpful during these iterations

Restructuration example

```
def encode(...) = {
 require(...)
 if remaining then
   if rle then
     ...
   else
     if otherRLE then
       ...

     if dictionary then
       ...
     else if diff then
       ...
     else
       ...
   assert(...)
   encode(...)
}.ensuring(...)
```

Restructuration example

```
def encode(...) = {
 require(...)
 if remaining then
   if rle then
     ...
   else
     if otherRLE then
       ...

     if dictionary then
       ...
     else if diff then
       ...
     else
       ...
   assert(...)
   encode(...)
}.ensuring(...)
```

Main encoding logic

Restructuration example

```
def encode(...) = {
 require(...)
 if remaining then
   if rle then
     ...
   else
     if otherRLE then
       ...

     if dictionary then
       ...
     else if diff then
       ...
     else
       ...
   assert(...)
   encode(...)
}.ensuring(...)
```

Express properties about encoding

Restructuration example

```
def encode(...) = {
 require(...)
 if remaining then
   if rle then
     ...
   else
     if otherRLE then
       ...

     if dictionary then
       ...
     else if diff then
       ...
     else
       ...
   assert(...)
   encode(...)
}.ensuring(...)
```

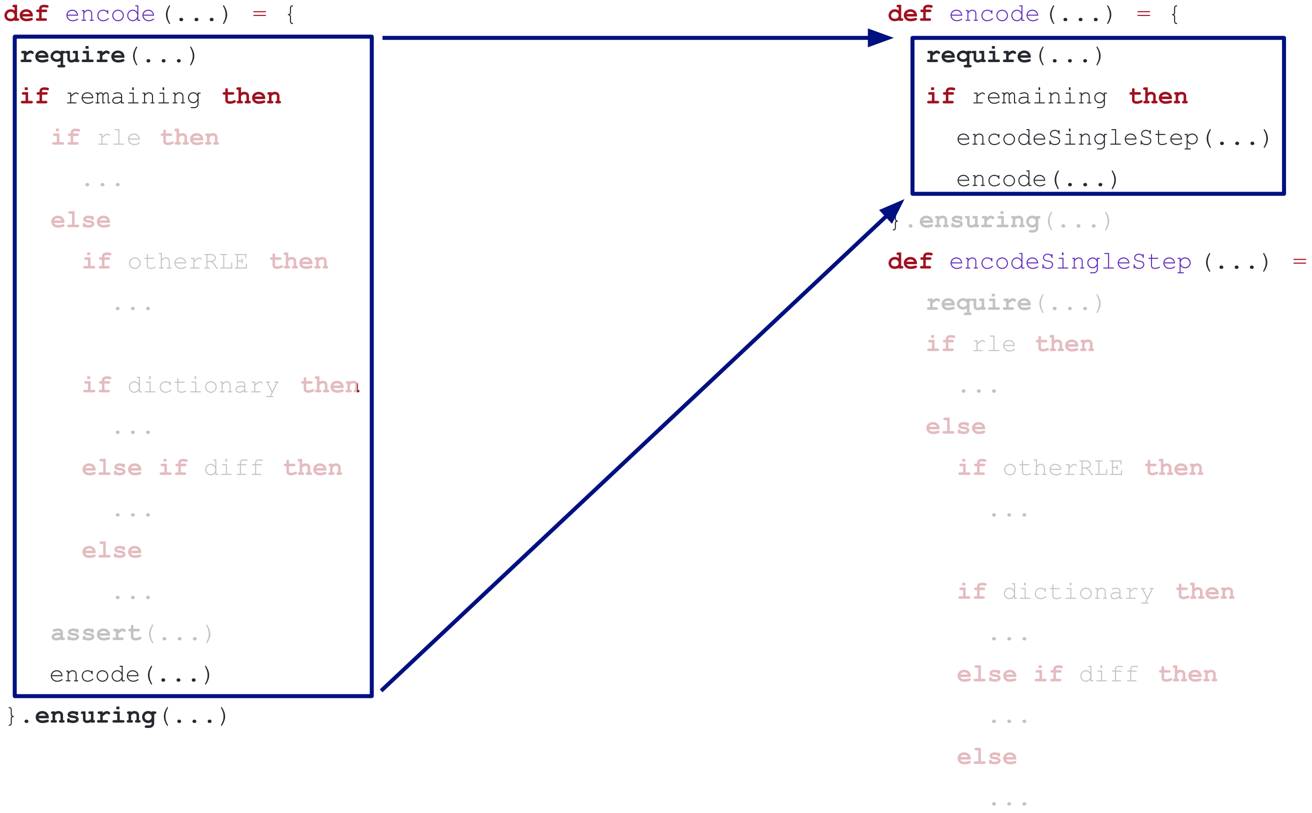
Implementation details result in huge VCs
Postcondition is too hard to prove!

Restructuration example

```
def encode(...) = {
 require(...)
 if remaining then
  if rle then
    ...
  else
    if otherRLE then
      ...

    if dictionary then
      ...
    else if diff then
      ...
    else
      ...
    assert(...)
    encode(...)
}.ensuring(...)
```

```
def encode(...) = {
  require(...)
  if remaining then
    encodeSingleStep(...)
    encode(...)
}.ensuring(...)
def encodeSingleStep(...) =
  require(...)
  if rle then
    ...
  else
    if otherRLE then
      ...

    if dictionary then
      ...
    else if diff then
      ...
    else
      ...
```

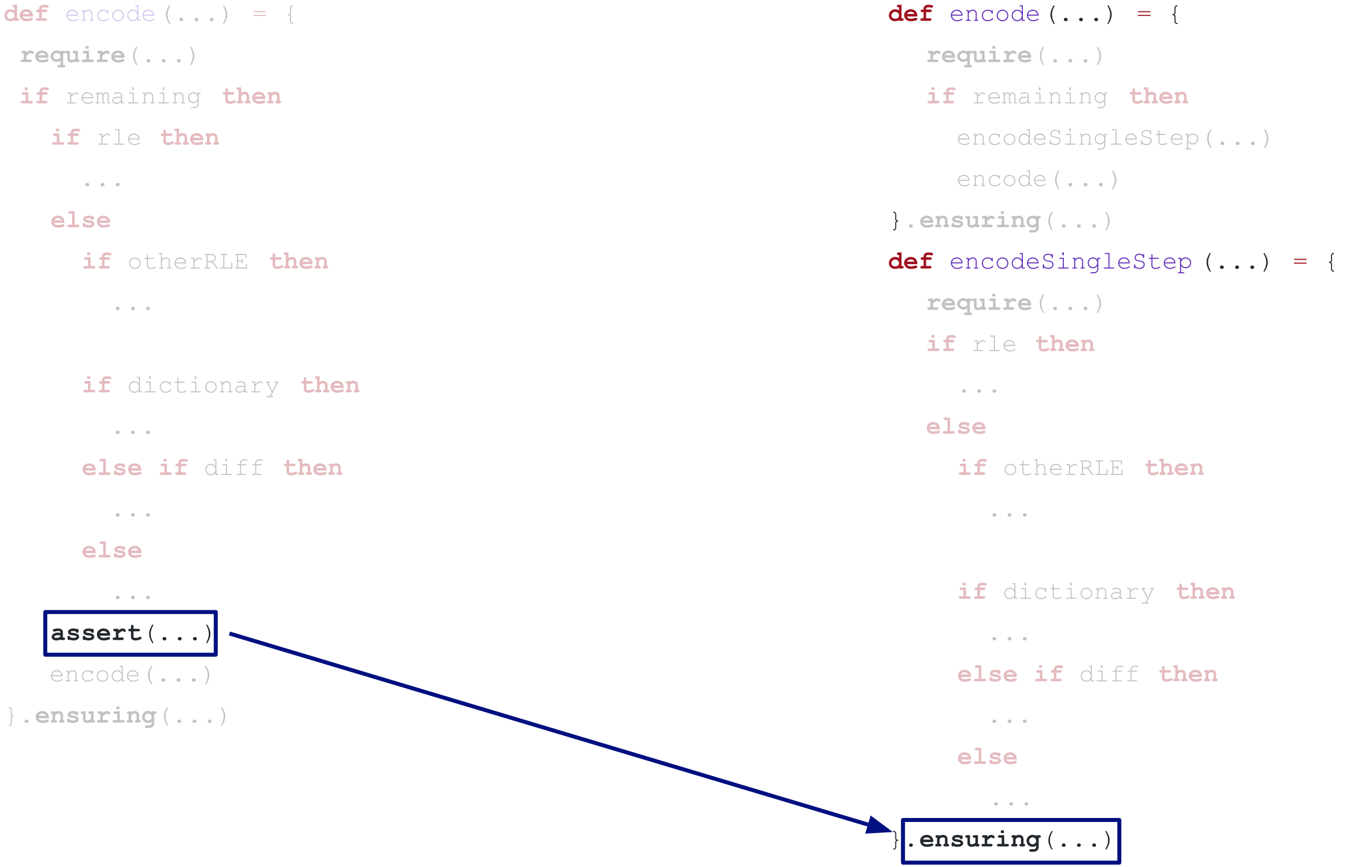
Restructuration example



```
def encode(...) = {
require(...)
if remaining then
  if rle then
    ...
  else
    if otherRLE then
      ...

    if dictionary then
      ...
    else if diff then
      ...
    else
      ...
  assert(...)
  encode(...)
}.ensuring(...)
```

```
def encode(...) = {
require(...)
if remaining then
  encodeSingleStep(...)
  encode(...)
}.ensuring(...)
def encodeSingleStep (...) =
  require(...)
  if rle then
    ...
  else
    if otherRLE then
      ...

    if dictionary then
      ...
    else if diff then
      ...
    else
      ...
```

Restructuration example



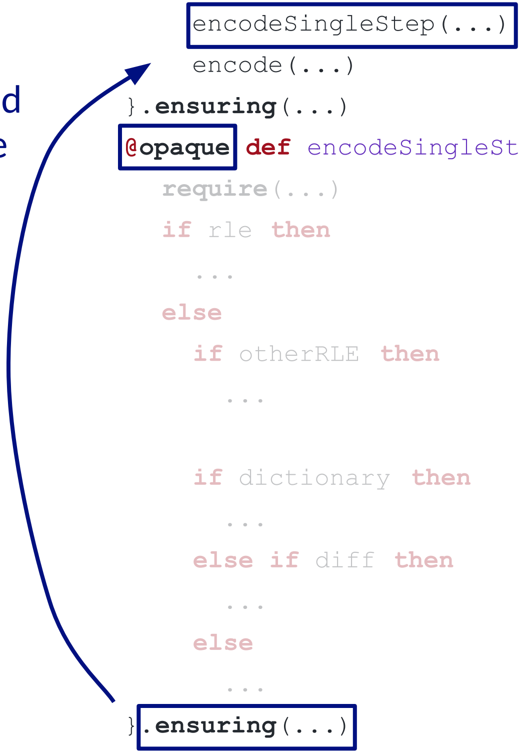
```
def encode(...) = {
 require(...)
 if remaining then
   if rle then
     ...
   else
     if otherRLE then
       ...

     if dictionary then
       ...
     else if diff then
       ...
     else
       ...
   assert(...)
   encode(...)
}.ensuring(...)
```

```
def encode(...) = {
  require(...)
  if remaining then
    encodeSingleStep(...)
    encode(...)
}.ensuring(...)
def encodeSingleStep(...) = {
  require(...)
  if rle then
    ...
  else
    if otherRLE then
      ...

    if dictionary then
      ...
    else if diff then
      ...
    else
      ...
}.ensuring(...)
```

Restructuration example

```
def encode(...) = {
 require(...)
 if remaining then
   if rle then
     ...
   else
     if otherRLE then
       ...

     if dictionary then
       ...
     else if diff then
       ...
     else
       ...
   assert(...)
   encode(...)
}.ensuring(...)
```

Only the core, desired properties are visible



```
def encode(...) = {
  require(...)
  if remaining then
    encodeSingleStep(...)
    encode(...)
}.ensuring(...)
@opaque def encodeSingleStep(...) = {
  require(...)
  if rle then
    ...
  else
    if otherRLE then
      ...

    if dictionary then
      ...
    else if diff then
      ...
    else
      ...
}.ensuring(...)
```

Restructuration example

```
def encode(...) = {
 require(...)
 if remaining then
   if rle then
     ...
   else
     if otherRLE then
       ...

     if dictionary then
       ...
     else if diff then
       ...
     else
       ...
   assert(...)
   encode(...)
}.ensuring(...)
```

Do the same for RLE

```
def encode(...) = {
  require(...)
  if remaining then
    encodeSingleStep(...)
    encode(...)
}.ensuring(...)
@opaque def encodeSingleStep(...) = {
  require(...)
  if rle then
    ...
  else
    if otherRLE then
      ...

    if dictionary then
      ...
    else if diff then
      ...
    else
      ...
}.ensuring(...)
```

Take away

- The main efforts are in:
    - Structuring the implementation to ease verification
    - Abstracting away details to describe high-level properties

Verification statistics

- Without proof code, our Scala implementation is 313 LOC
    - Against 311 for the C reference
- With proof code, it reaches 2789 LOC
    - Of which 1405 are dedicated to lemmas
- 42 lemmas, of which 19 are general-purpose
- 3591 Verification Conditions (VCs)
- ~50 mins to run on a 20-cores server
- 66% of VCs are dedicated to checking preconditions and 22% to assertions

C code generation with Stainless

- The implementation happens to follow the C codegen restrictions
- Ghost code (contracts, assertion) is erased
- Generated C code has 661 LOC (against 311 for the reference)
- With -O3, the generated code is on-par with the reference for both encoding and decoding
  - Modern C compilers are amazing :)

	Decoding [MP/s]	Encoding [MP/s]
Reference	90.92	86.24
Transpiled	97.65	84.45

Final words

- QOI is a simple yet practical image compression algorithm
- We proved its correctness with Stainless
  - Implementation adaptation and restructuration helped in that regard
- The transpiled C code exhibits similar performance as the reference
  - Verified code does not need to compromise over performance
- Stainless project: https://github.com/epfl-lara/stainless
- QOI Case Study: https://github.com/epfl-lara/bolts/tree/master/qoi

Stainless

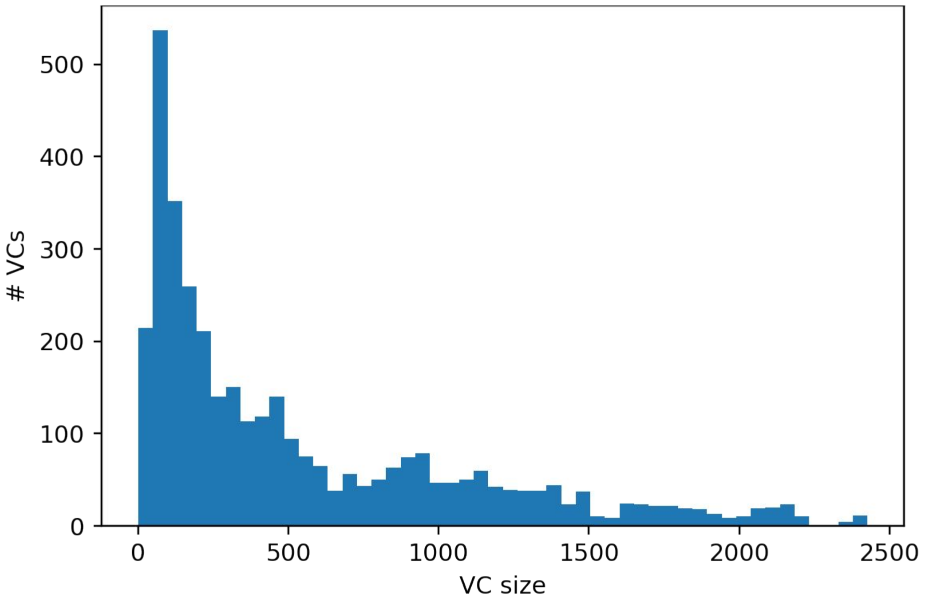
QOI

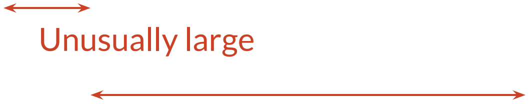
FIRST-AID SLIDES

OPEN IN CASE OF WICKED QUESTIONS

THIS DECK IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED SUCCESS

VCs (tree) size distribution




Unusually large

Very large

Bounds requirements

```
def inputInv(pixels: Array[Byte], w: Long, h: Long, chan: Long): Boolean =
  0 < w && w <= MaxWidth &&
  0 < h && h <= MaxHeight &&
  3 <= chan && chan <= 4 &&
  w * h * chan == pixels.length
```

Invertibility?

```
def encode(img: Array[Byte]): Array[Byte] = img

def decode(data: Array[Byte]): Array[Byte] = img

def bigBrain(img: Array[Byte]): Boolean = {
  decode(encode(img)) == img
}.holds
```

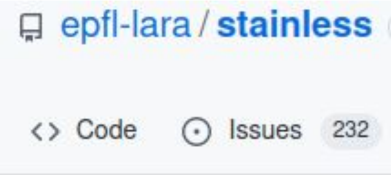
- Oh no, there goes our contribution :(

Invertibility?

```
def encode(img: Array[Byte]): Array[Byte] = img

def decode(data: Array[Byte]): Array[Byte] = img

def bigBrain(img: Array[Byte]): Boolean = {
  decode(encode(img)) == img
}.holds
```

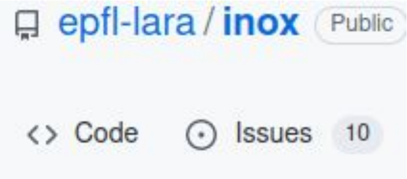
- ~~Oh no, there goes our contribution :(~~
- This solution does not adhere to QOI specifications
- Can we be certain ours does?
- No, but we can be sure data is never lost by the implemented compression (whether or not it follows the QOI format)

On the trustworthiness of Stainless



epfl-lara / stainless

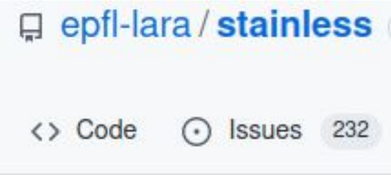
<> Code ⊙ Issues 232

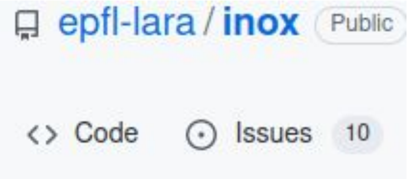


epfl-lara / inox Public

<> Code ⊙ Issues 10

On the trustworthiness of Stainless





- It is true the trust we put may not always be justified
- Nevertheless, it increases the confidence we have, more than testing would do
  - Though tests are always welcome!