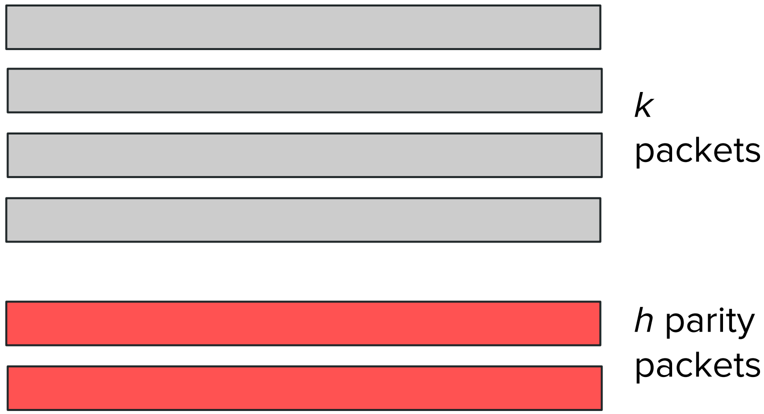
Verified Erasure Correction in Coq with MathComp and VST

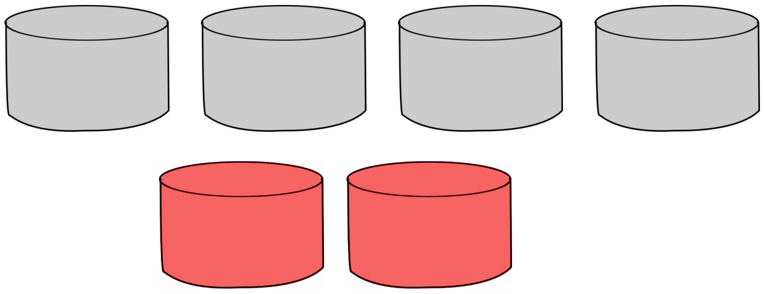
Josh Cohen
Princeton University
8/9/2022
With Qinshi Wang and Andrew Appel

Error-Correcting Codes

- When we send data over network, some may not arrive
- In some cases, retransmission infeasible or impossible
    - low latency applications, satellite communications, RAID
- Solution: add additional "parity" packets/bits and reconstruct of lost data
- Parities chosen using Error-Correcting Code
- Lots of ECCs exist (Hamming, Reed-Solomon, Convolutional, BCH, etc), most based on fairly sophisticated math
- Correctness is difficult to formally prove



k disks

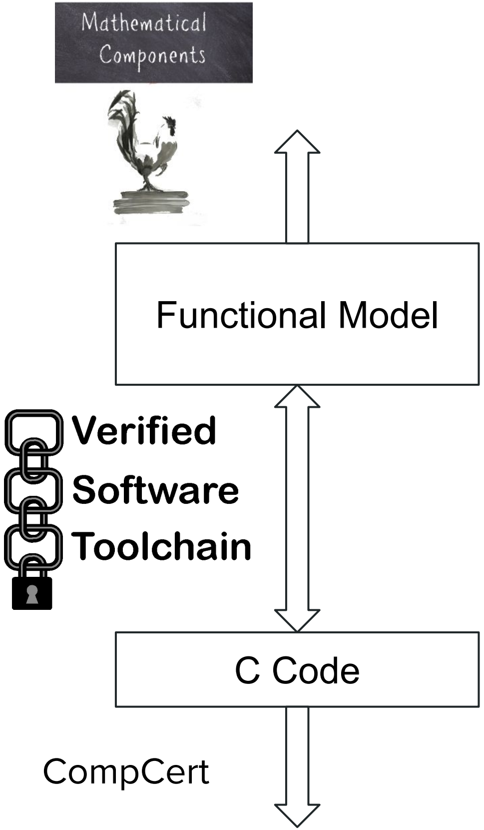


h parity disks

Project Goals

- Formally verify real-world C implementation of FEC with Coq and the Verified Software Toolchain (VST)
- C code was originally written by Anthony McAuley of Bellcore in ‘90s, in active use since
- Algorithm is modified Reed-Solomon, developed by Rabin [Journal of the ACM 1989], McAuley [SIGCOMM 90], and others
  - Includes unpublished optimizations, correctness unknown to authors
- Intriguing target for verification
  - Need to connect high-level correctness with low-level implementation
  - Algorithm based on finite fields, polynomials, linear algebra, low level uses clever C programming tricks

Verification Overview



- *Layered verification* - separate proofs with a *functional model*
- CompCert (Leroy) - C compiler written and verified in Coq
- VST (Appel) - C program logic and proof automation
  - Proved sound wrt CompCert C
- Mathematical Components - large library of formalized math
  - Ex: groups, rings, fields, matrices, polynomials + theorems
- Very different ecosystem, types, tactics
  - Unclear if VST+MathComp could be used together

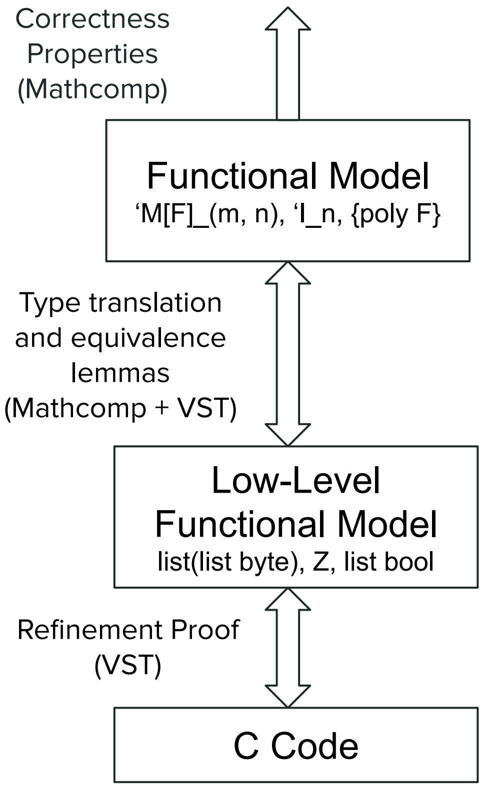
Reed-Solomon Coding

- Interpret data as a polynomial over a finite field
  - ie: $(a_0, a_1, \ldots, a_{k-1}) \rightarrow a_0 + a_1 x + \ldots + a_{k-1} x^{k-1}$
- Evaluate polynomial at *k+h* distinct points in the field
- Equivalently, multiply by Vandermonde matrix
- To make systematic, multiply by row-reduced Vandermonde matrix
- Decoder is a bit complicated, but not as bad as full Reed-Solomon
- Will be able to recover data if receive at least *k* packets of *k+h* total

$$\begin{bmatrix} a_0 & a_1 & a_2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 \\ x_0 & x_1 & x_2 \\ x_0^2 & x_1^2 & x_2^2 \end{bmatrix}$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & 0 & x_1 & x_2 & x_3 \\ 0 & 1 & 0 & x_4 & x_5 & x_6 \\ 0 & 0 & 1 & x_7 & x_8 & x_9 \end{pmatrix}$$

Verification Details



- We really need 2 functional models
  1. Define high-level functional model with MathComp types
  2. Prove correctness properties of functional model (MathComp/Coq)
  3. Define low-level functional model with VST/CompCert types and prove equivalence
  4. Prove that C code refines low-level functional model (VST)
- Allows us to use VST and Mathcomp together

Verification Example - Gaussian elimination

- Standard algorithm in linear algebra to row reduce a matrix over a field
  - transform using row swaps, scalar multiplication, and adding multiples of rows
- Can be used to calculate inverses, determinants, solve systems of linear equations
- In this application - used to create weight matrix and invert matrix in decoder

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} \end{pmatrix} \to \begin{pmatrix} 1 & 0 & 0 & x_1 & x_2 & x_3 \\ 0 & 1 & 0 & x_4 & x_5 & x_6 \\ 0 & 0 & 1 & x_7 & x_8 & x_9 \end{pmatrix}$$

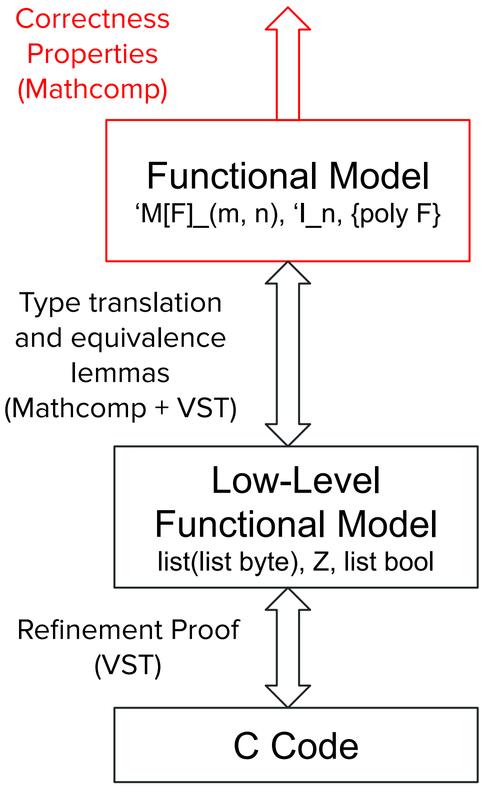
Verification Example - Gaussian elimination

```c
int fec_matrix_transform (fec_sym * p, fec_sym i_max, fec_sym j_max) {

  fec_sym *n, *m, *q, *r, inv;
  fec_sym i, j, k, w;

  for (k = 0; k < i_max; k++) {

    for (i = 0; i < i_max; i++){
      q = (p + (i * j_max) + j_max - 1);
      m = q - j_max;
      w = i;

      while (*(q - k) == 0) {

        if (++w == i_max){
          return (FEC_ERR_TRANS_FAILED);
        }

        if (*(p + (w * j_max) + j_max - 1 - k) != 0){
          printf ("FEC: swap rows (not done yet!)\n");
          return (FEC_ERR_TRANS_SWAP_NOT_DONE); /* Not done yet! */
        }
      }

      inv = fec_invefec[*(q - k)];

      for (n = q; n > m; n--) {
       *n = FEC_GF_MULT (*n, inv);
      }
    }

    r = (p + (k * j_max) + j_max - 1);
    for (i = 0; i < i_max; i++) {

      if (i != k) {
        q = (p + (i * j_max) + j_max - 1);

        for (j = 0; j < j_max; j++) {
          *(q - j) = *(q - j) ^ (*(r - j));
        }
      }
    }
  }

  for (i = 0; i < i_max - 1; i++) {
    q = (p + (i * j_max) + j_max - 1);
    m = q - j_max;
    inv = fec_invefec[*(q - i)];

    for (n = q; n > m; n--) {
      *n = FEC_GF_MULT (*n, inv);
    }
  }
  return (0);
}
```

```
Definition gaussian_elim {m n} (A: 'M[F]_(m, n)) :=
  all_lc_1 (gauss_all_steps A (insub 0%N) (insub 0%N)).
```

$$\left[\ A \mid I\ \right] \xrightarrow[\text{elim}]{\text{Gaussian}} \left[\ I \mid A^{-1}\ \right]$$

Verification Example - Gaussian elimination



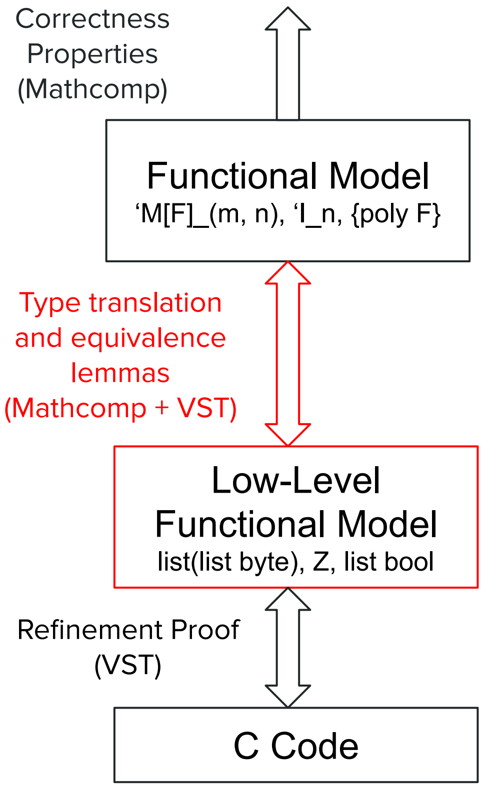
1. Define functional model and prove correctness properties

```
Definition gaussian_elim {m n} (A: 'M[F]_(m, n)) :=
  all_lc_1 (gauss_all_steps A (insub 0%N) (insub 0%N)).
```

$$\left[\; A \;\middle|\; I \;\right] \xrightarrow[\text{elim}]{\text{Gaussian}} \left[\; I \;\middle|\; A^{-1} \;\right]$$

```
Definition find_invmx {n} (A: 'M[F]_n) :=
  rsubmx (gaussian_elim (row_mx A 1%:M)).


Lemma gaussian_finds_invmx: forall {n} (A: 'M[F]_(n, n)),
  A \in unitmx ->
  find_invmx A = invmx A.
```

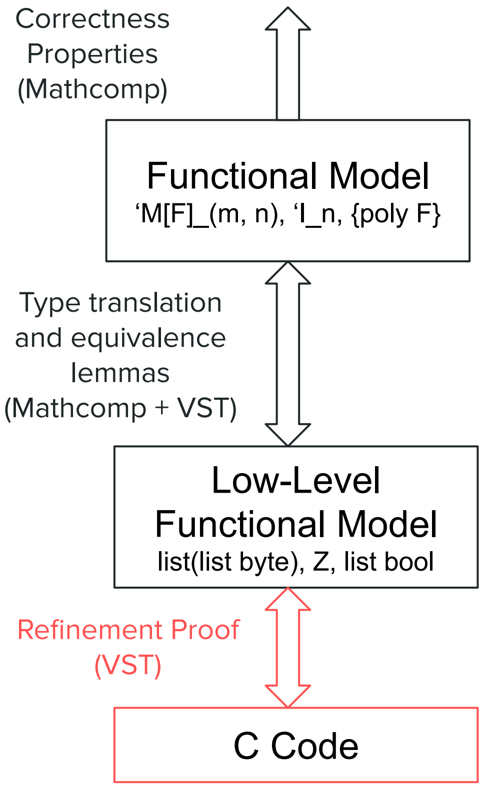
Verification Example - Gaussian elimination



2. Define low-level functional model and prove equivalence

```
Definition lmatrix := list (list byte).

Definition gauss_restrict_list m n (mx: lmatrix) :=
all_lc_one_partial m n (gauss_all_steps_list_partial m n mx m) (m-1).

Lemma gauss_restrict_list_equiv: forall {m n} (mx: lmatrix) (Hmn: m <= n),
  wf_lmatrix mx m n ->
  lmatrix_to_mx m n (gauss_restrict_list m n mx) =
  gaussian_elim_restrict_noop (lmatrix_to_mx m n mx) (le_Z_N Hmn).
```

Verification Example - Gaussian elimination



3. Define and prove VST spec using low-level functional model

```
int fec_matrix_transform (fec_sym * p, fec_sym i_max, fec_sym j_max)
```

```
Definition fec_matrix_transform_spec :=
  DECLARE _fec_matrix_transform
  WITH gv: globals, m : Z, n : Z, mx : list (list byte), s : val, sh: share
  PRE [ tptr tuchar, tuchar, tuchar]
    PROP (0 < m <= n; n <= Byte.max_unsigned; wf_lmatrix mx m n;
          strong_inv_list m n mx; writable_share sh)
    PARAMS (s; Vubyte (Byte.repr m); Vubyte (Byte.repr n))
    GLOBALS (gv)
    SEP (FIELD_TABLES gv;
         data_at sh (tarray tuchar (m * n)) (map Vubyte (flatten_mx mx)) s)
  POST [tint]
    PROP()
    RETURN (Vint Int.zero)
    SEP(FIELD_TABLES gv;
        data_at sh (tarray tuchar (m * n))
          (map Vubyte (flatten_mx (gauss_restrict_list m n mx))) s).
```

```
Lemma body_fec_matrix_transform : semax_body Vprog Gprog
  f_fec_matrix_transform fec_matrix_transform_spec.
```

Challenge - Restricted Gaussian Elimination

- C code implements “restricted” Gaussian elimination
  - no swaps, assumes all elements in current column are nonzero
- Only works of all elements in rth column are nonzero!
- C code returns errors if this condition is violated
  - “FEC: swap rows (not done yet!)”
- Suggests that authors were unclear whether this was sufficient

$$\begin{pmatrix} a & 0 & b & c \\ 0 & d & e & f \\ 0 & 0 & g & h \\ 0 & 0 & i & j \end{pmatrix} \to \begin{pmatrix} \frac{a}{b} & 0 & 1 & \frac{c}{b} \\ 0 & \frac{d}{e} & 1 & \frac{f}{e} \\ 0 & 0 & 1 & \frac{h}{g} \\ 0 & 0 & 1 & \frac{j}{i} \end{pmatrix} \to \begin{pmatrix} \frac{a}{b} & 0 & 0 & \frac{c}{b} - \frac{h}{g} \\ 0 & \frac{d}{e} & 0 & \frac{f}{e} - \frac{h}{g} \\ 0 & 0 & 1 & \frac{h}{g} \\ 0 & 0 & 0 & \frac{j}{i} - \frac{h}{g} \end{pmatrix}$$

```
while (*(q - k) == 0) {          /* if zero */
    if (++w == i_max) {
        return (FEC_ERR_TRANS_FAILED);  /* failed */
    }
    if (*(p + (w * i_max) + i_max - 1 - k) != 0){         /* swap rows */
        printf ("FEC: swap rows (not done yet!)\n");
        return (FEC_ERR_TRANS_SWAP_NOT_DONE);   /* Not done yet! */
    }
}
```

Challenge - Restricted Gaussian Elimination

- Determined and proved in Coq: Restricted Gaussian Elimination equal to full Gaussian Elimination iff a certain m^2 submatrices (for $m \times n$ matrix) are all invertible
    - VERY strong condition - does not hold of identity, diagonal, triangular, etc
- In this application: run Gaussian Elim on Vandermonde matrix and submatrices of row-reduced Vandermonde matrix
- Property holds of these matrices (nontrivially) due to properties of Vandermonde matrices and polynomials

Verifying the C Code

- Difficult to verify - written over 25 years ago, never designed to be verified
- One challenge: represents matrices as 2D global arrays, partially-filled 2D local arrays, 1D arrays, pointers, and pointer to array of pointers
  - Need lemmas and tactics to convert between these, added to VST
- Found 1 bug

Bug in Implementation

```
q = (p + (i * j_max) + j_max - 1);
m = q - j_max;
for (n = q; n > m; n--) {
        //loop body
}
```

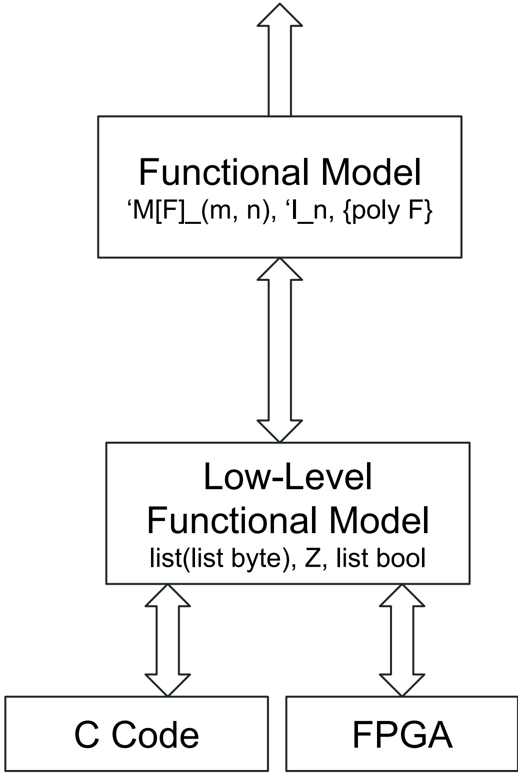
- In loop; when i=0, m points to p-1
- n > m is undefined behavior!
- VST will not let us prove this program correct without modifying it
- VST gives strong guarantees about program behavior - no undefined behavior, no extra IO/system calls/etc

Related Work

- In **Network Function Verification**, VigNAT [Zaostrovnykh et al., SIGCOMM 2017], Vigor [Zaostrovnykh et al., SOSP 2019], and Gravel [Zhang et al., NSDI 2020] use more automated methods to verify NAT, load balancer, firewall, and more, but have restrictions on state and cannot handle things like unbounded loops
- Various **Error-Correcting Codes** have been formalized in Coq [Affeldt et al., Journal of Automated Reasoning 2020 and others], Lean [Hagiwara et al., ISITA 2015 and Kong et al., ISITA 2018], and ACL2 [Nasser et al., Journal of Electronic Testing 2020]
- Our work is the first to connect a sophisticated ECC with a real-world, efficient implementation

Conclusion and Future Work

- Core FEC code is fully verified (https://github.com/verified-network-toolchain/Verified-FEC)
- Ongoing - code that handles buffer and packet management (calls core FEC code)
  - Specification is much more difficult - need to deal with streams of packets and define usable spec
- Possible future work - implement incremental FEC encoding and decoding at line rate on an FPGA, verify correctness according to same functional model
- Other future projects connecting MathComp and VST (numerical methods)



Questions?

Thanks for listening!