Verifying the POCKET+ lossless compression algorithm

Milestone 3

June 2026 - Project in Formal Methods in Programming

Cláudia Faria PG60240
Patrícia Bastos PG60287

Main Objective

Formal verification of the Pocket+ protocol

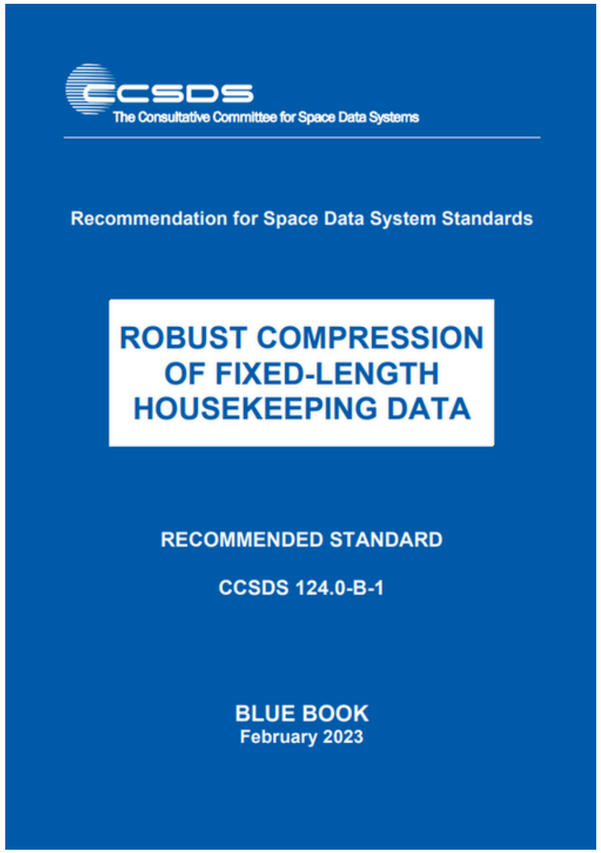
TELESPAZIO
a LEONARDO and THALES company

This project is an international collaboration with the space company **Telespazio**.

Milestone 3

The protocol

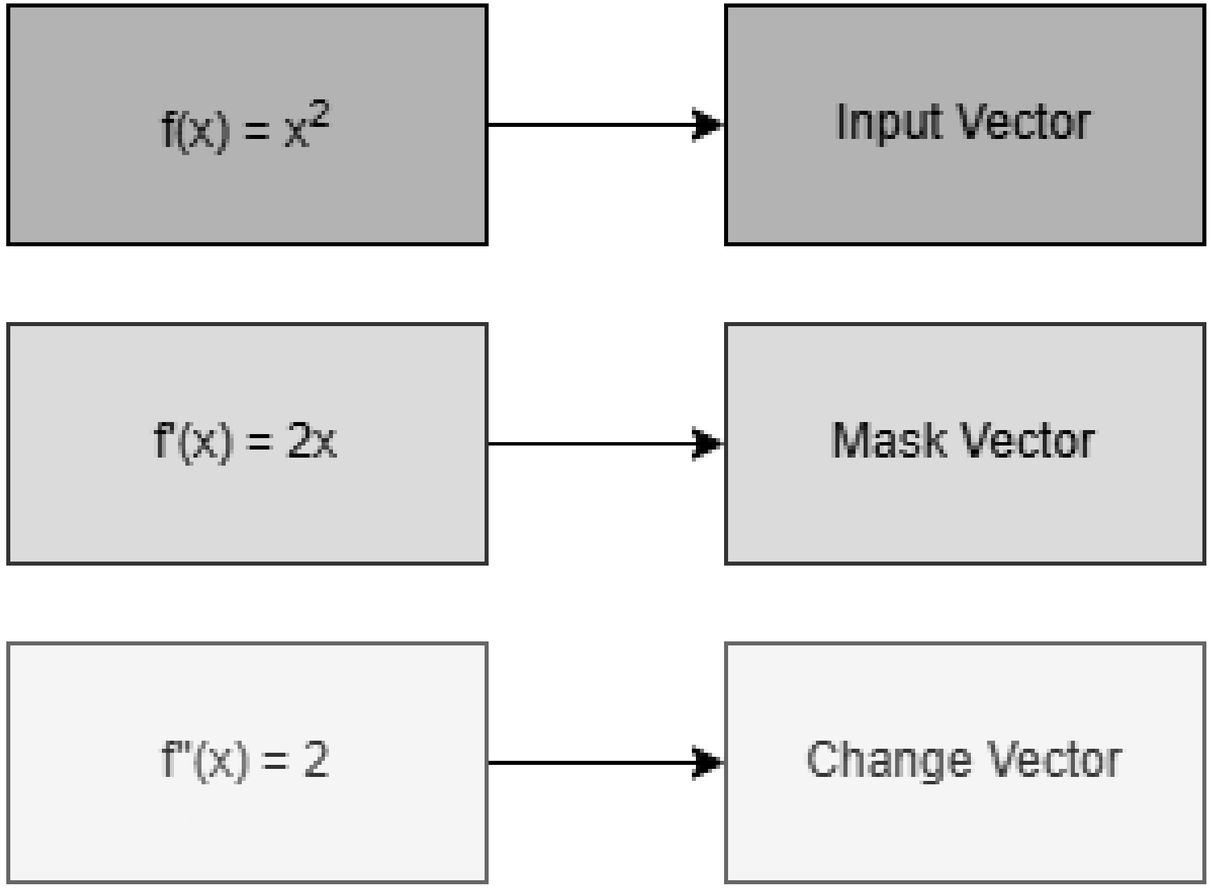
The Recommended Standard



The core of the protocol

Flags	Determine the protocol’s behaviour
Mask Update	Identifies predictable bits
RLE	Packet encoding function
Resynchronization	Restores information “lost”

The core of the protocol



Parameters

Mask Update

New Mask Flag (p_t)

- Always user-defined bit.

- Indicates whether the Mask will be replaced, enabling bit positions to change classification from unpredictable to predictable if they have not changed state since the last time.

Encoding

Send Mask Flag (f_t)

- Shall be 1 for each time index t, where the value of t is lower than the current R_t. Otherwise, it is a user-defined bit.

- Causes the entire mask vector to be encoded in the output vector when it is set to one.

Uncompressed Flag (r_t)

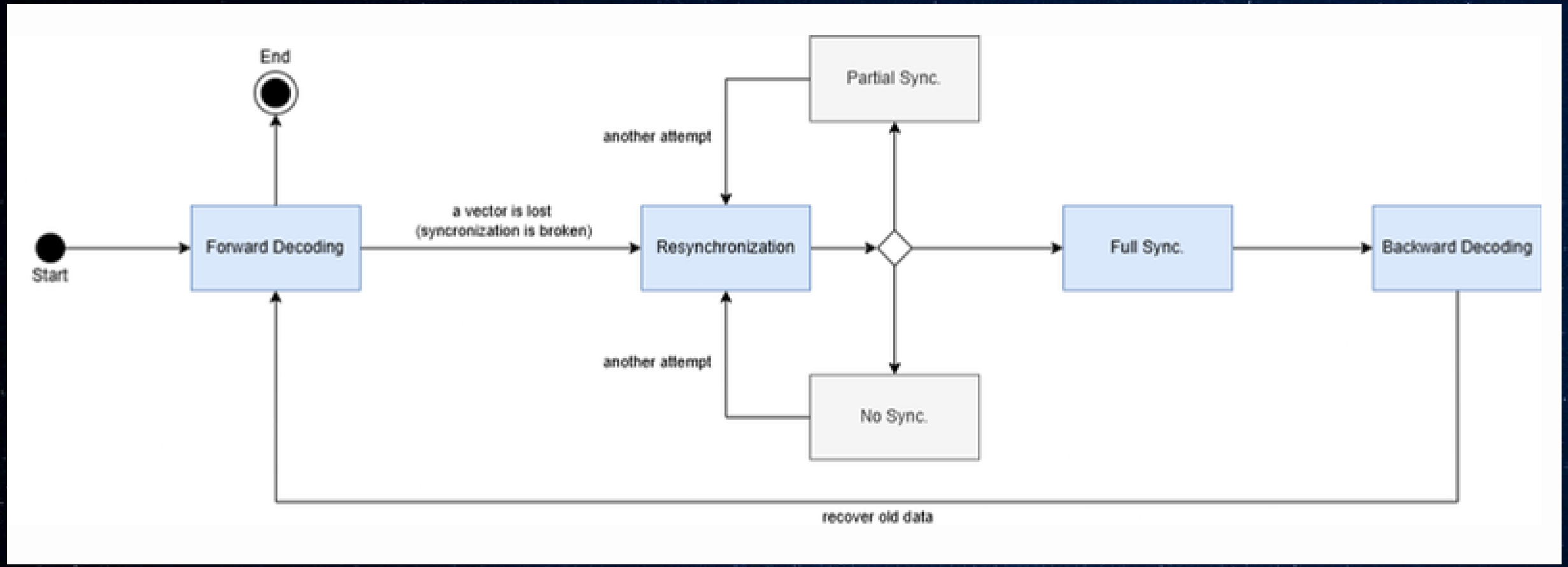
- Shall be 1 for each time index t, where the value of t is lower than the current R_t. Otherwise, it is a user-defined bit.

- Causes the entire input vector to be included in the output vector when it is set to one.

Minimum Required Effective Robustness Level (R_t)

- Always user-specified integer between 0 and 7 at each time index t.

- Controls the guaranteed number of consecutive output vectors that can be lost prior to the current output vector without affecting the ability to decompress it.


End
Partial Sync.
another attempt
Start
Forward Decoding
a vector is lost
(syncronization is broken)
Resynchronization
Full Sync.
Backward Decoding
another attempt
No Sync.
recover old data

Milestone 3

Implementation

Existing Implementation

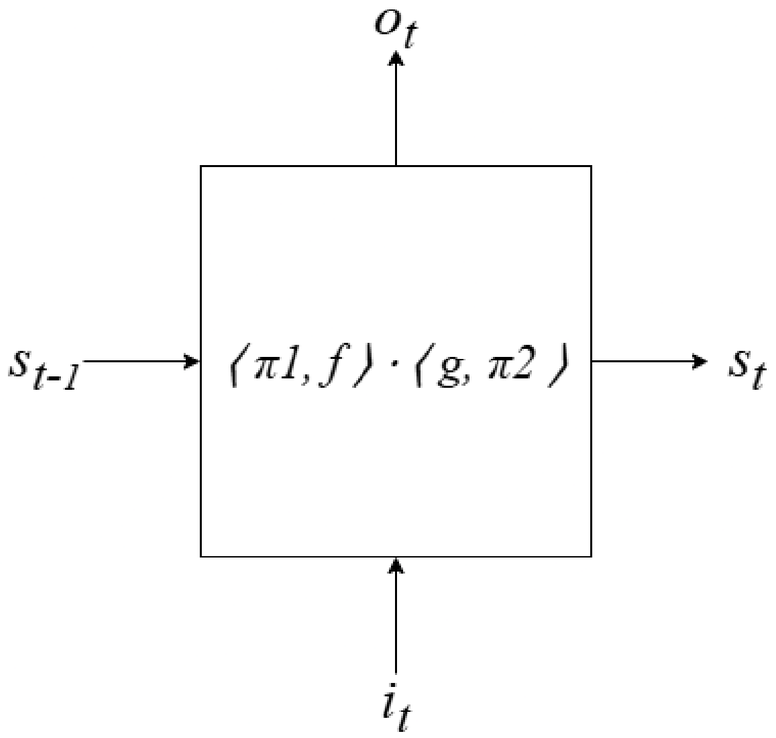
C++ code implemented by

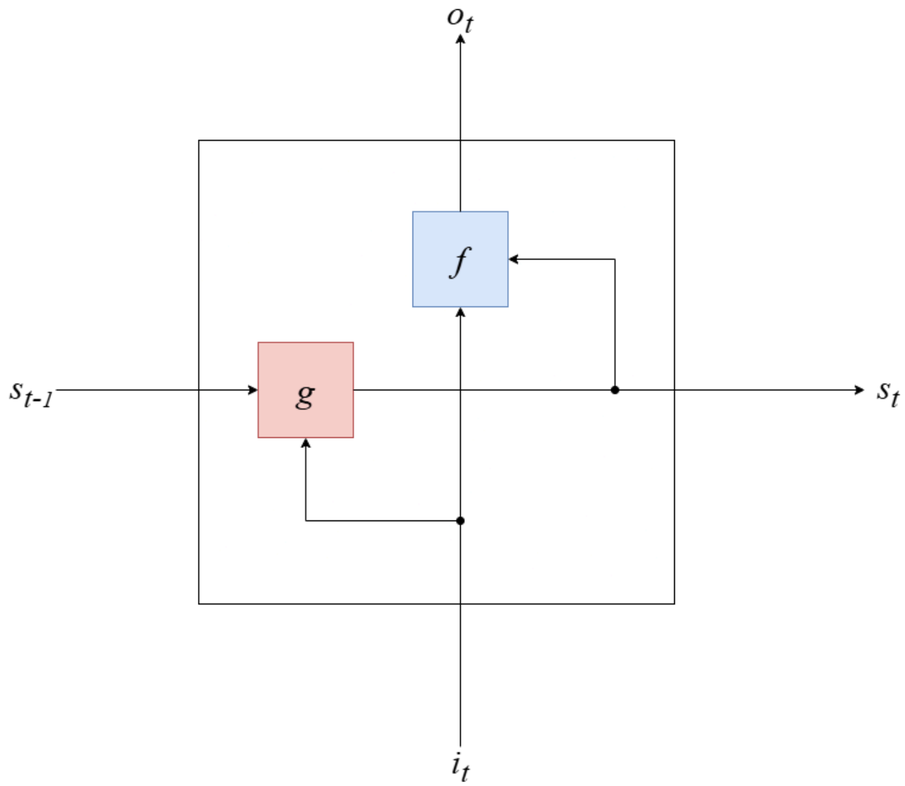
VISION SPACE

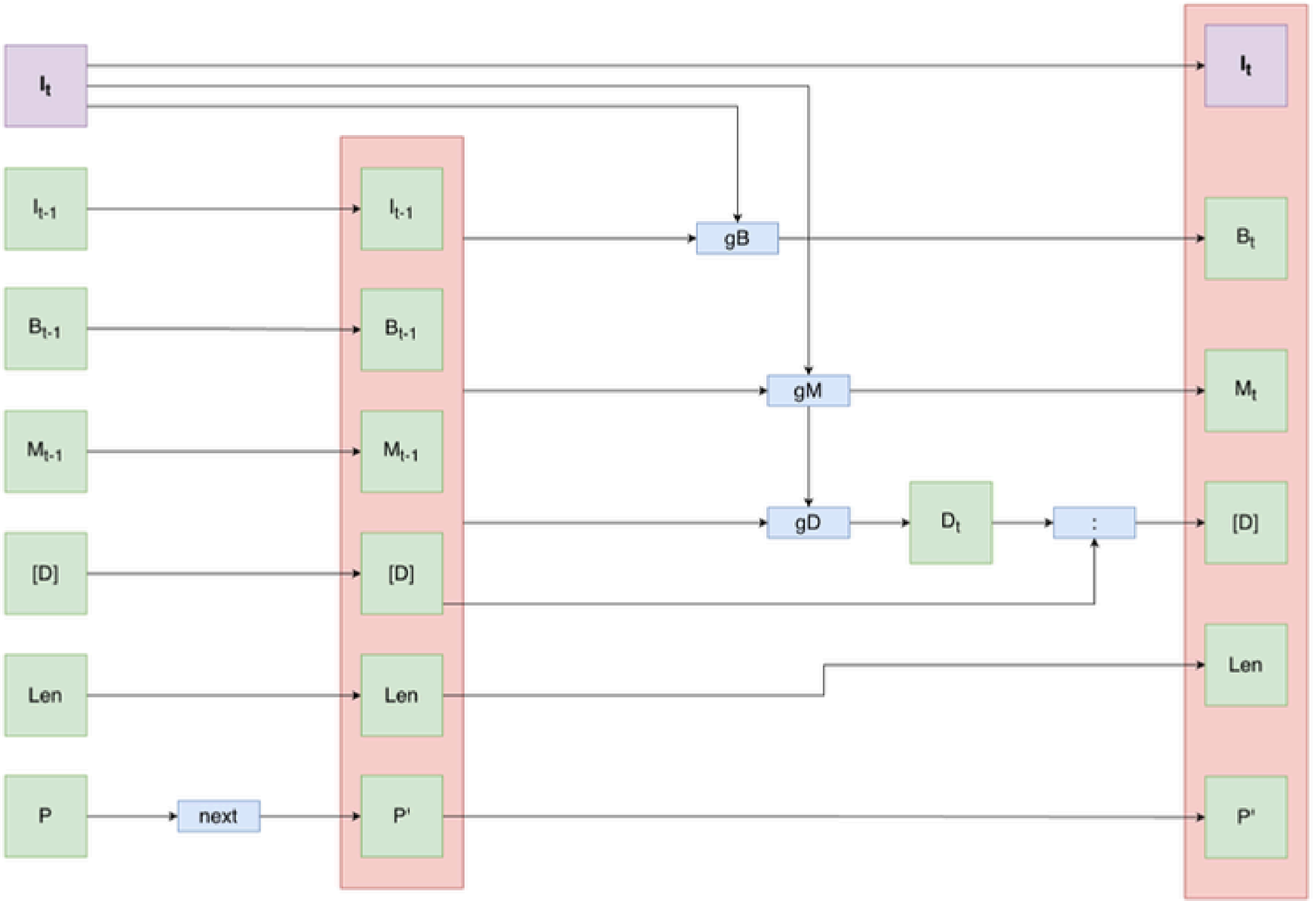
Existing Implementation

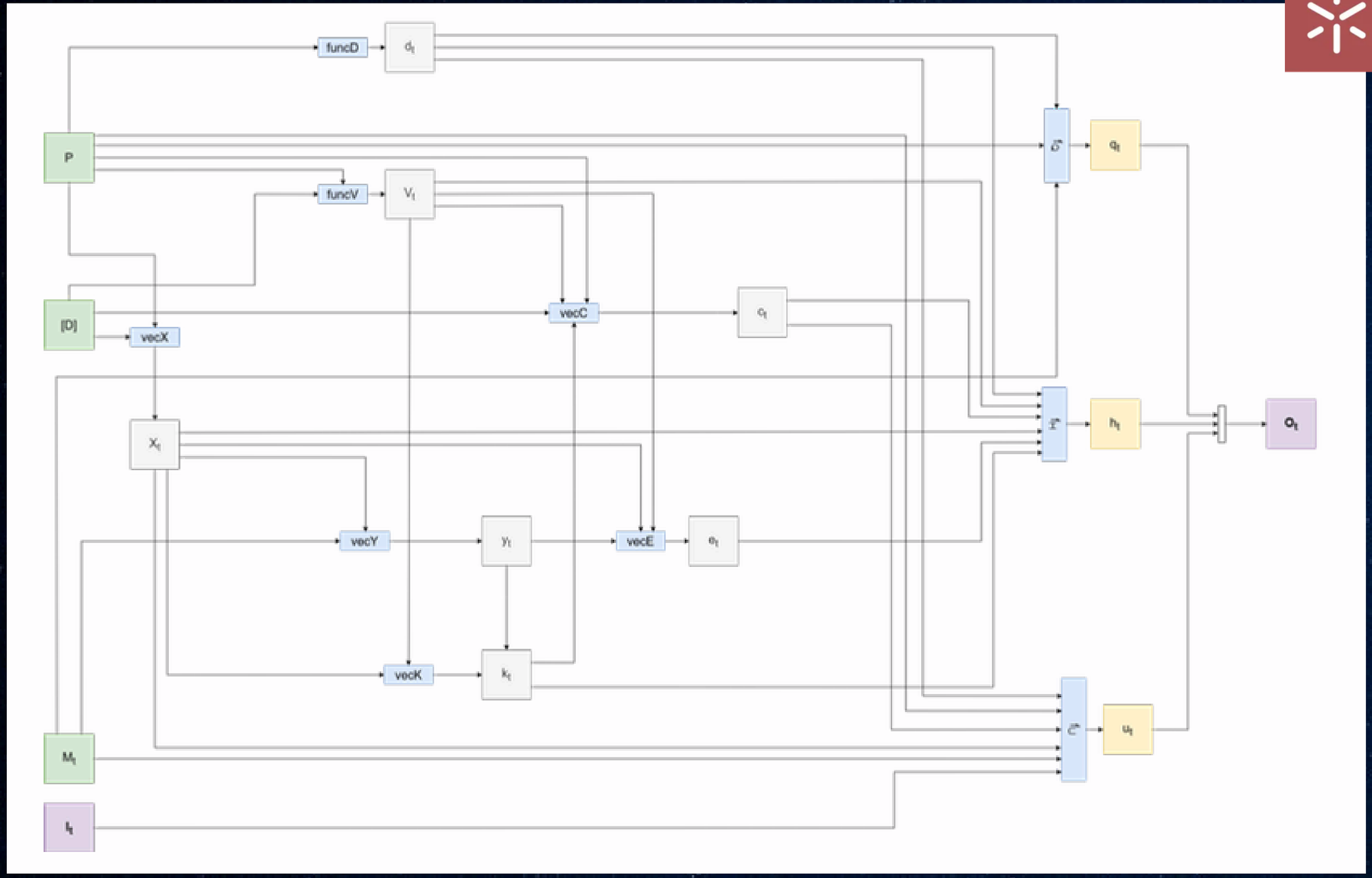
```
// Revert COUNT operation
if(*it == 0){
    out.emplace_back(1);
    it += 1;
    in.pop_front();
}
else if(((it + 1) != in.end()) && ((it + 2) != in.end()) && ((it + 3) != in.end()) && ((it + 4) != in.end())){
    if((*it == 1) && (*(it+1) == 1) && (*(it+2) == 0)){
        it += 3;
        pocketplus::utils::pop_n_from_front(in, 3);
        // Undo BIT_5(A - 2)
        if((in.size() < 5) || in.empty()){
            throw std::invalid_argument("Revert of COUNT(X) failed");
        }
        auto count_tmp = std::make_unique<unsigned int>(0);
        auto bit_shift = std::make_unique<unsigned int>(0);
        for(auto ite = it + 4; ite >= it; ite--, *bit_shift += 1){
            if(*ite){
                *count_tmp |= 1 << *bit_shift;
            }
        }
```

Encoder - Mealy Machine chain

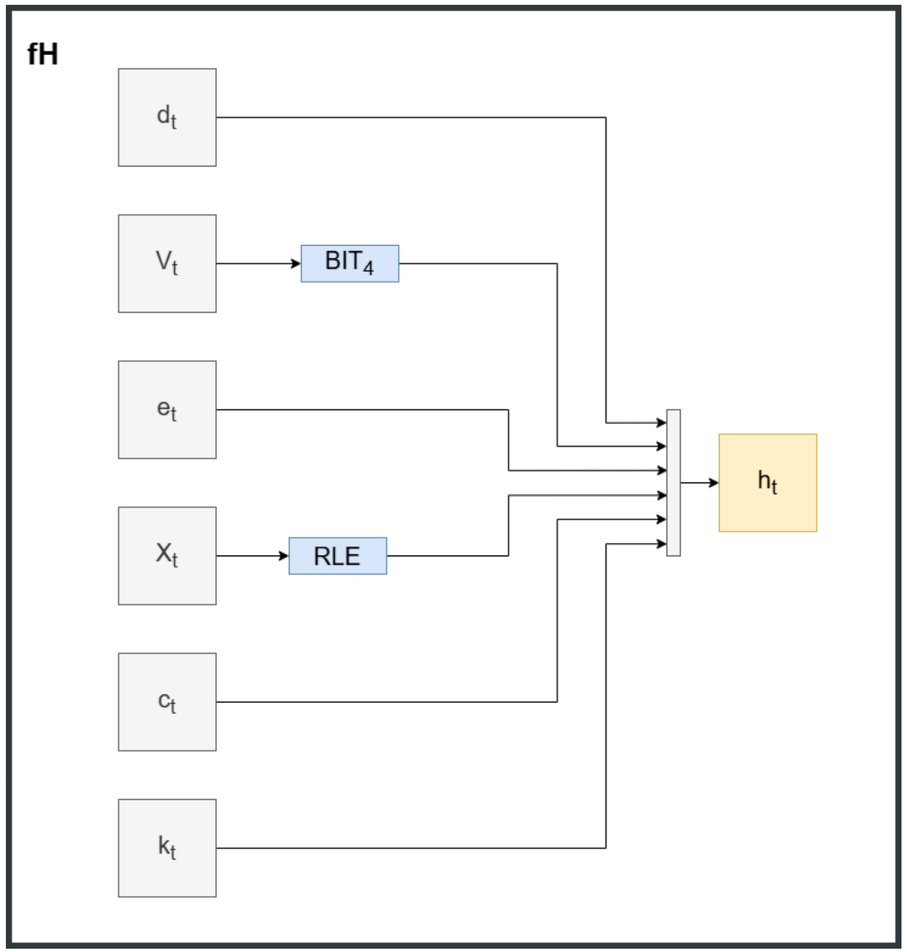





I_t
I_{t-1}
B_{t-1}
M_{t-1}
[D]
Len
P
next
I_{t-1}
B_{t-1}
M_{t-1}
[D]
Len
P'
gB
gM
gD
D_t
:
I_t
B_t
M_t
[D]
Len
P'

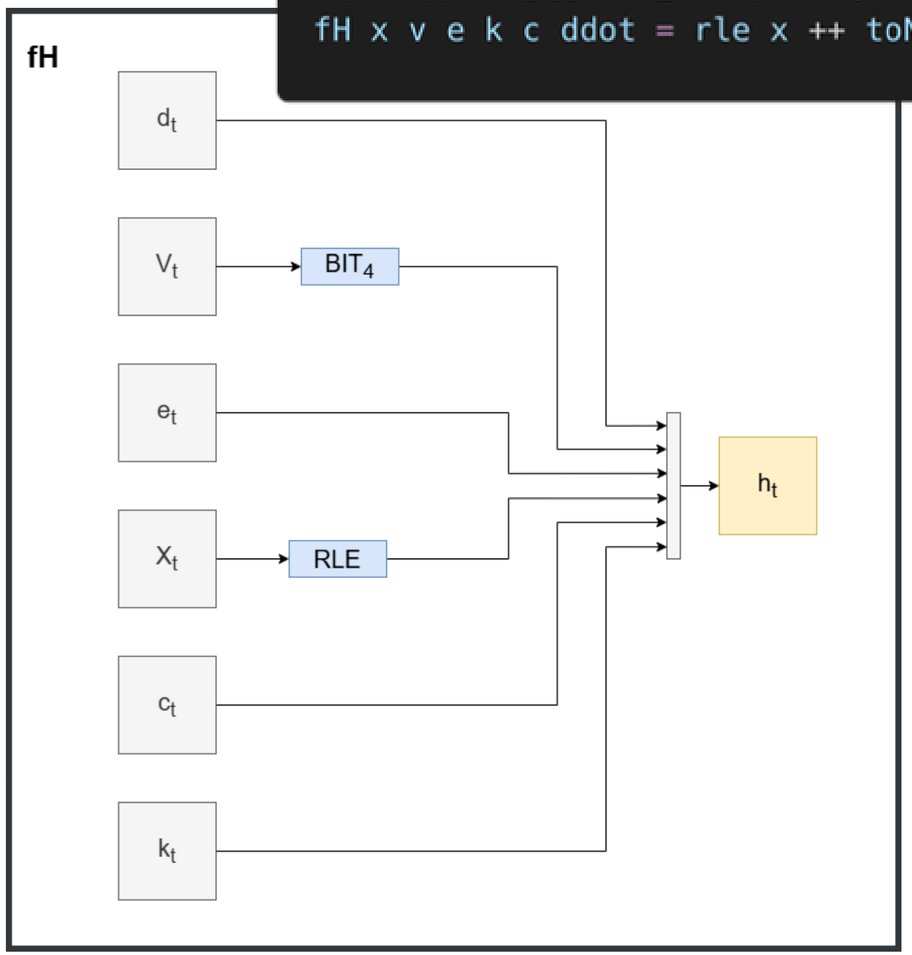

funcD
d_t
P
$\mathcal{D}$
q_t
funcV
V_t
[D]
vecX
vecC
c_t
$\mathcal{E}$
h_t
o_t
X_t
vecY
y_t
vecE
e_t
vecK
k_t
$\mathcal{C}$
u_t
M_t
I_t

Encoding step

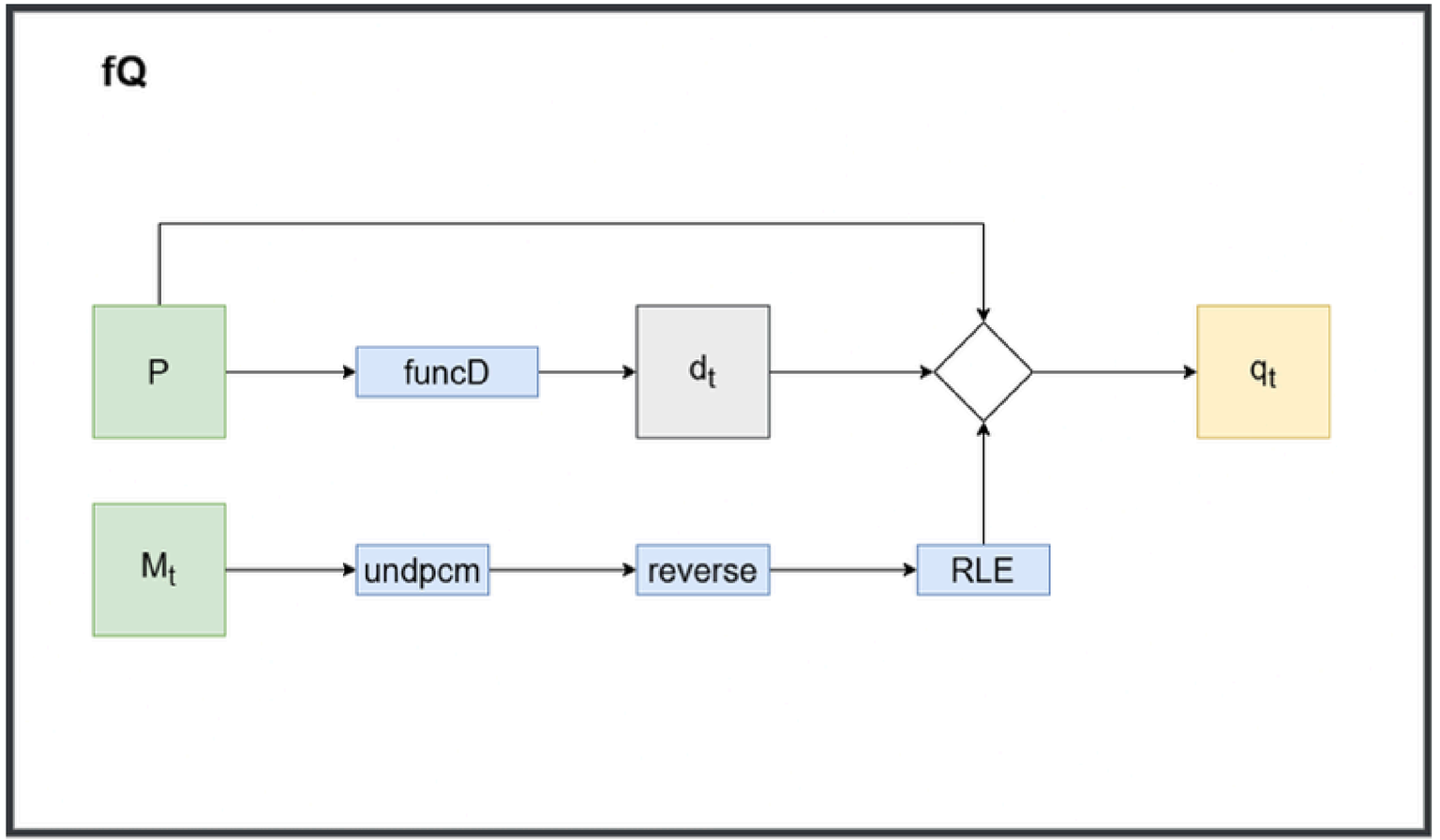


Encoding step

```
fH :: X -> Bit -> E -> K -> C -> Bit -> [Bit]
fH x v e k c ddot = rle x ++ toNBits 4 v ++ e ++ k ++ c ++ [ddot]
```

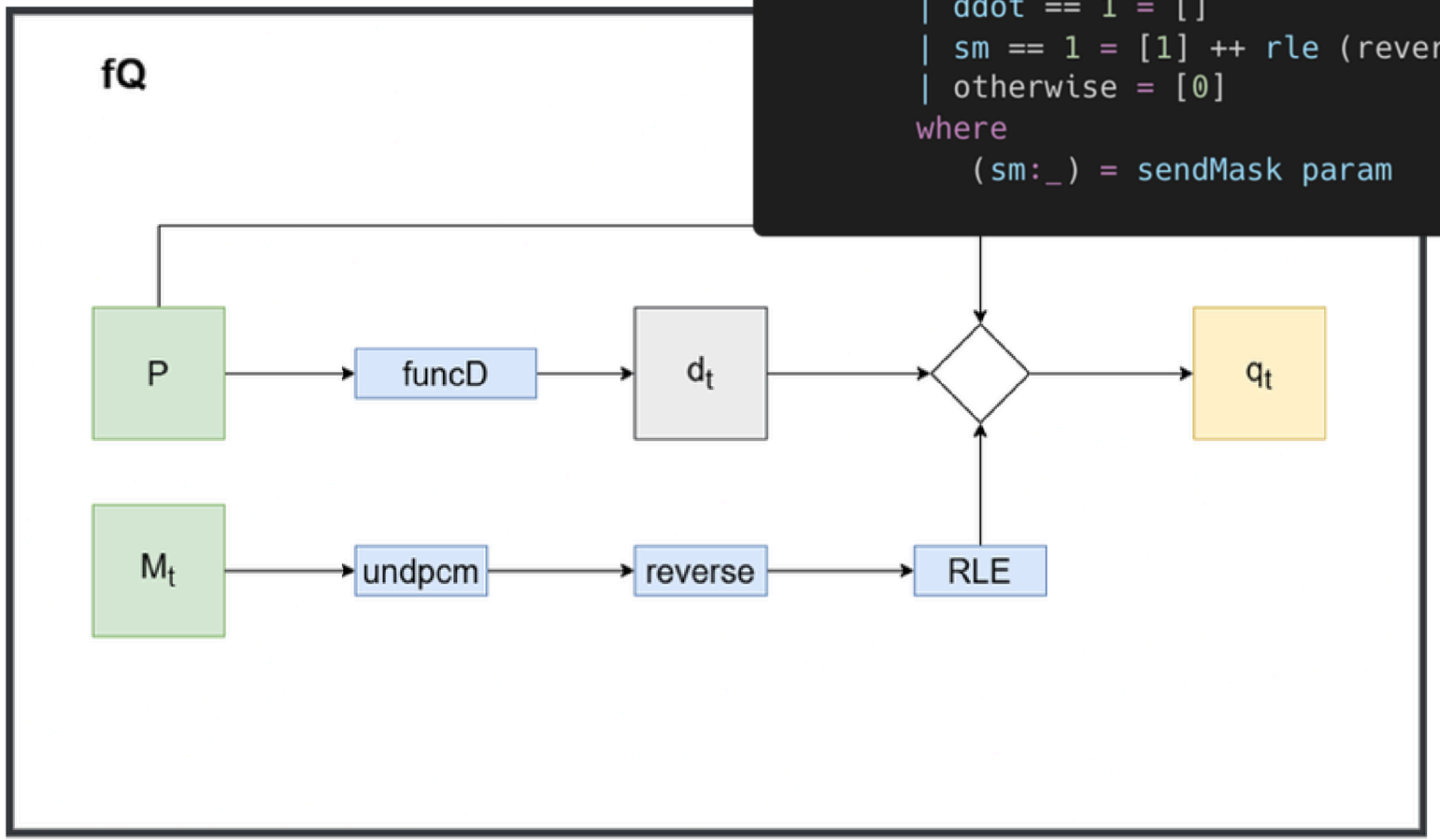


Encoding step

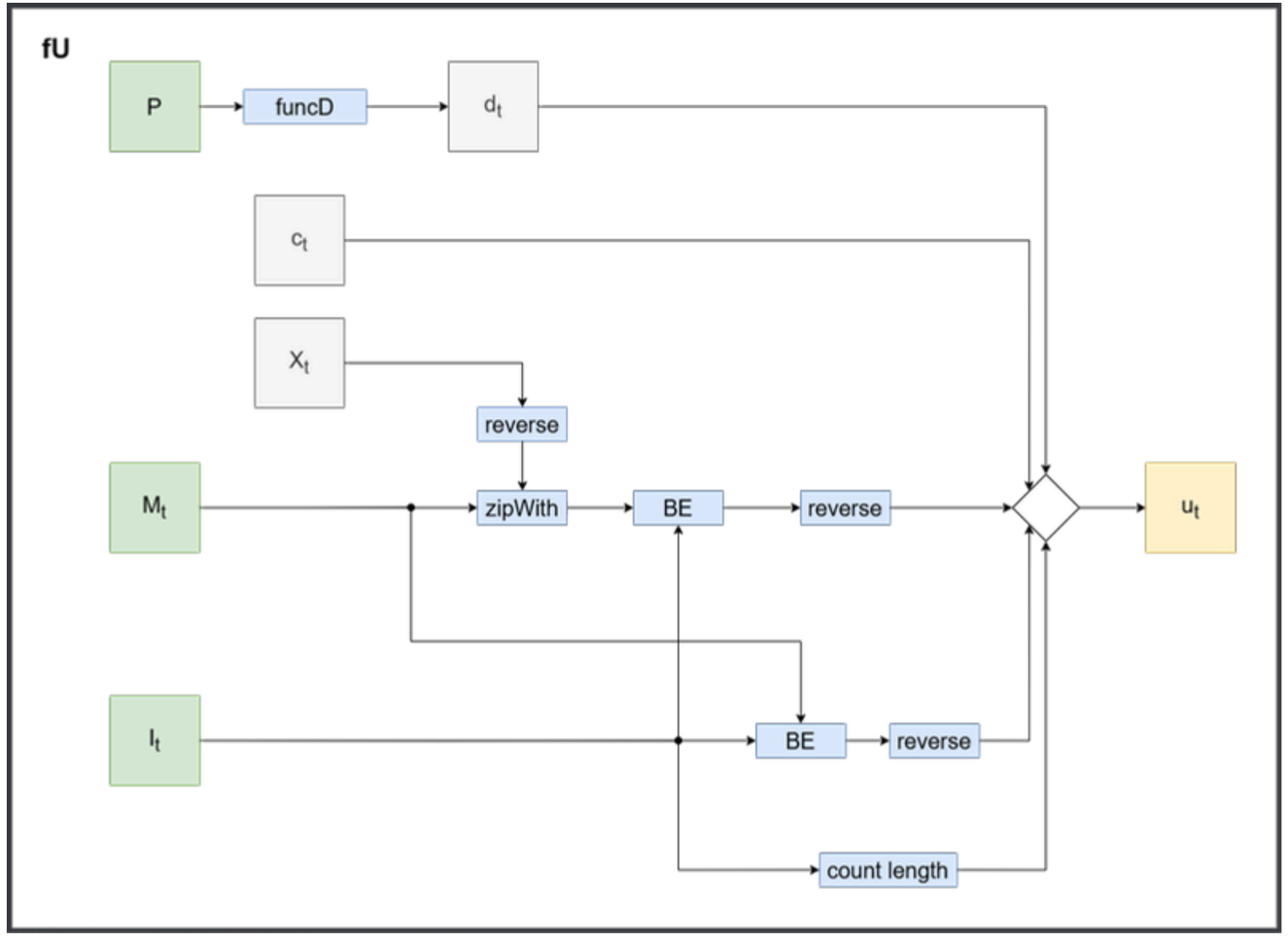


Encodin

```haskell
fQ :: S -> Bit -> [Bit]
fQ (_, _, m, _, _, param) ddot
        | ddot == 1 = []
        | sm == 1 = [1] ++ rle (reverse (undpcm m))
        | otherwise = [0]
        where
           (sm:_) = sendMask param
```

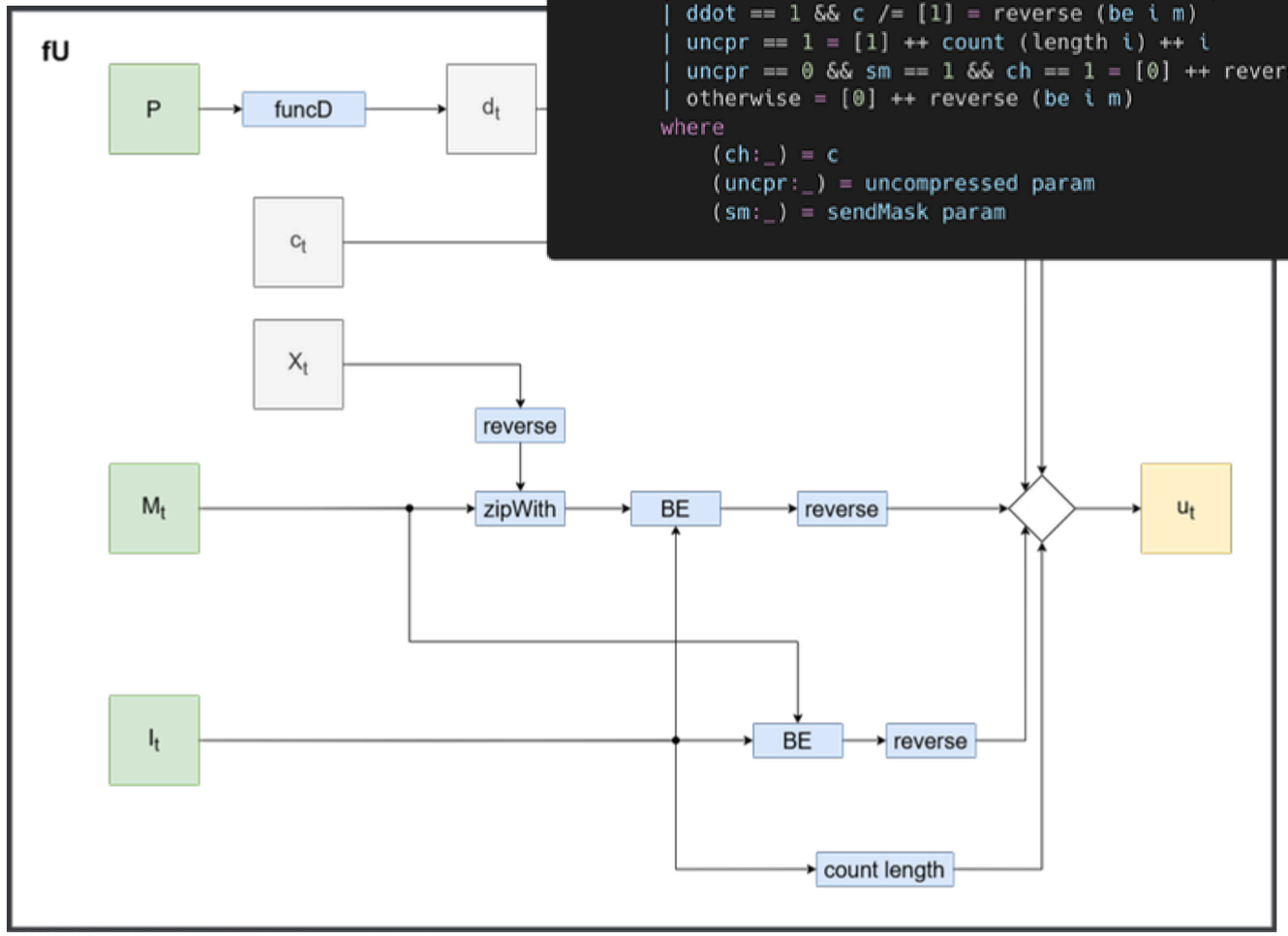


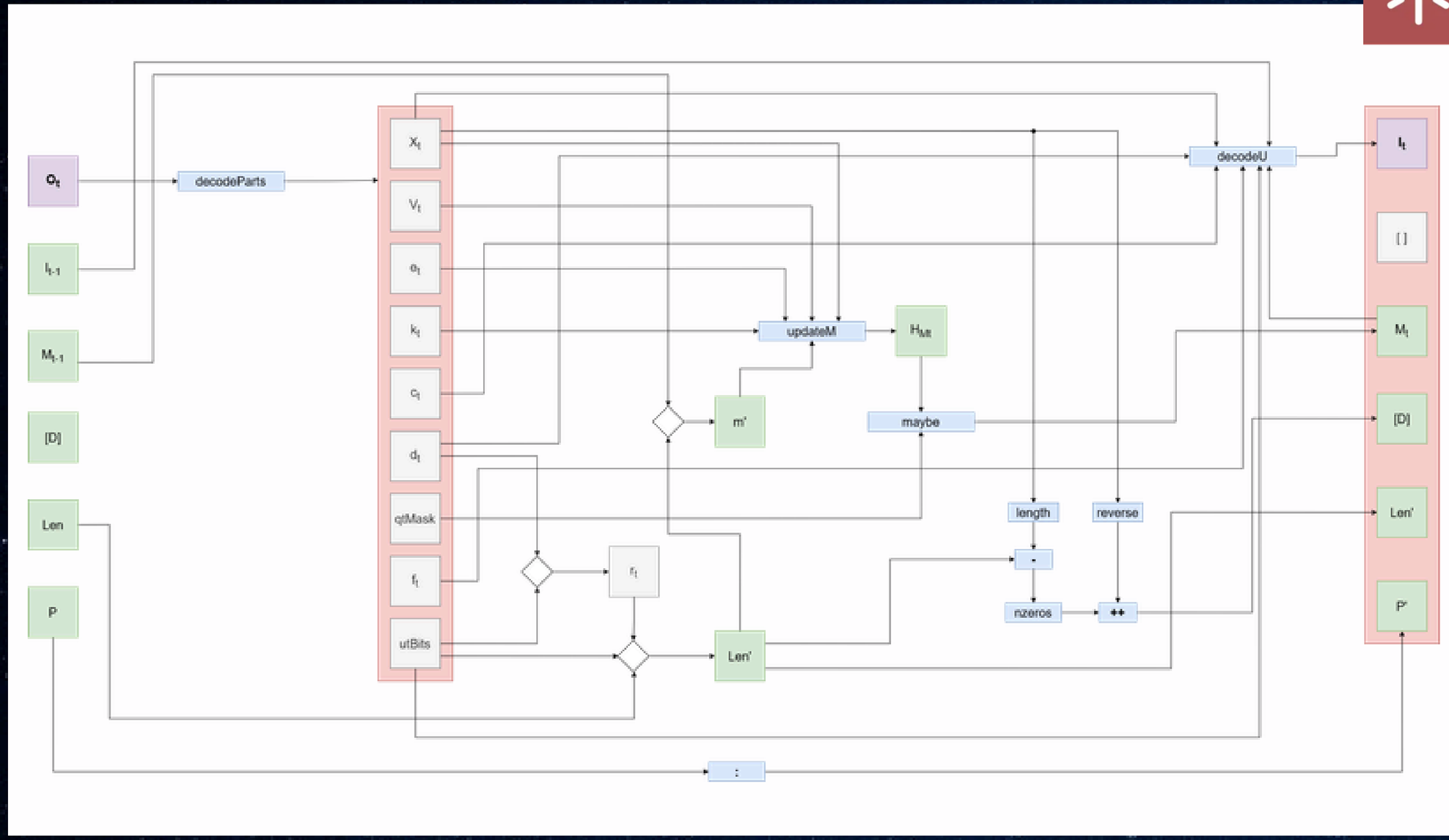
Encoding step



Encoc

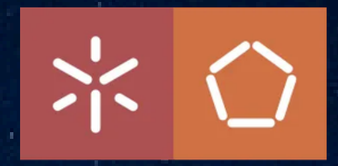
```haskell
fU :: I -> S -> X -> C -> Bit -> [Bit]
fU i (_, _, m, _, _, param) x c ddot
      | ddot == 1 && c == [1] = reverse (be i (zipWith (.|.) (reverse x) m))
      | ddot == 1 && c /= [1] = reverse (be i m)
      | uncpr == 1 = [1] ++ count (length i) ++ i
      | uncpr == 0 && sm == 1 && ch == 1 = [0] ++ reverse (be i (zipWith (.|.) (reverse x) m))
      | otherwise = [0] ++ reverse (be i m)
      where
          (ch:_) = c
          (uncpr:_) = uncompressed param
          (sm:_) = sendMask param
```




O_t
decodeParts
I_{t-1}
M_{t-1}
[D]
Len
P
X_t
V_t
o_t
k_t
c_t
d_t
qtMask
f_t
utBits
updateM
H_{kt}
m'
maybe
r_t
Len'
length
reverse
-
nzeros
++
decodeU
I_t
[ ]
M_t
[D]
Len'
P'
:

Milestone 3

Correctness



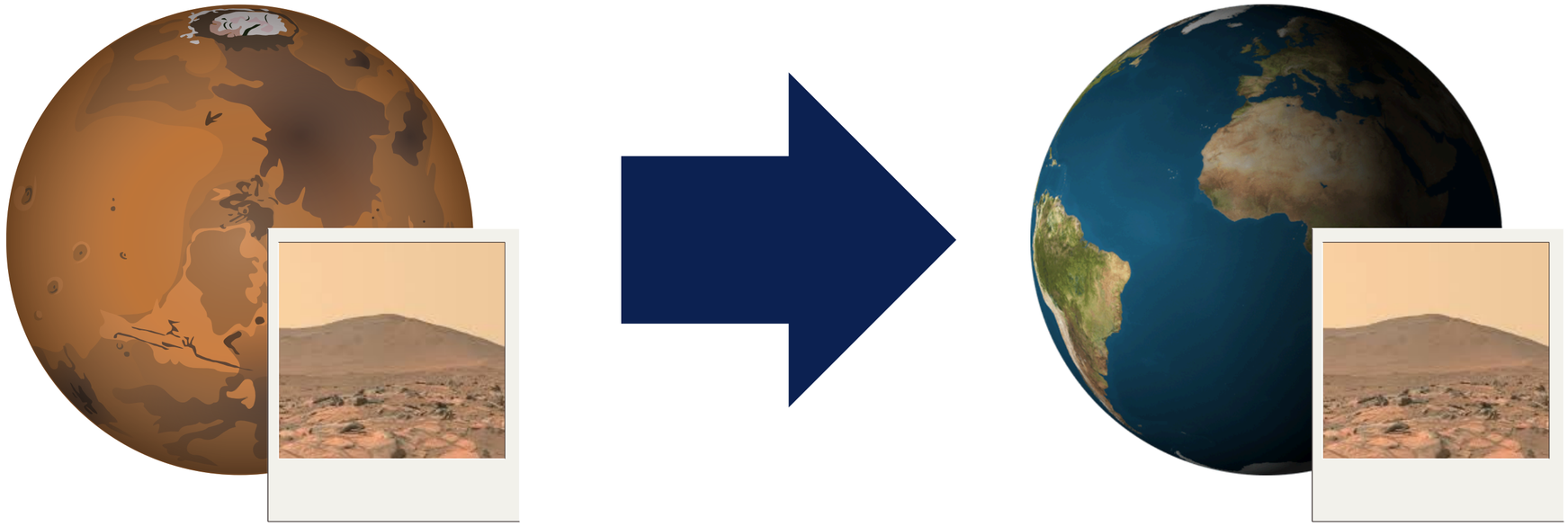
$$\text{Decoder} = \text{Encoder}^{\circ}$$

too strong

$$\text{Decoder} \cdot \text{Encoder} = \text{id}$$

exactly what we need

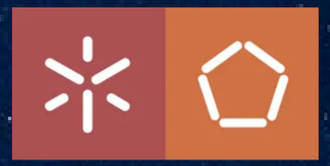
Reversibility



Minimal complements

A minimal complement of a function contains just enough information that was **originally missing** to make it **reversible.**

By attaching this minimal complement, the function becomes injective, and thus reversible.



Run-Length Encoding

RLE encoding:

```
rle :: [Bit] → [Bit]
rle = counting · init · map (+1) · map length · splitOn [1]
```

RLE decoding:

```
unrle :: [Bit] → Int → [Bit]
unrle = flip pipeline where
  pipeline n = intercalate [1] · map (flip replicate 0) · map (subtract 1) · trail n · uncounting
  trail n s = s ++ [n − (sum s + length s)]
```

Run-Length Encoding

Adding minimal complements...

$$rle' :: ([Bit] \to ([Bit], Int))$$
$$rle' = \langle rle, length \rangle$$

$$unrle' :: ([Bit], Int) \to [Bit]$$
$$unrle' = \widehat{unrle}$$

But we're not done yet.

RLE decomposed

RLE encoding:

```
rle :: [Bit] → [Bit]
rle = counting · init · map (+1) · map length · splitOn [1]
```

RLE decoding:

```
unrle :: [Bit] → Int → [Bit]
unrle = flip pipeline where
  pipeline n = intercalate [1] · map (flip replicate 0) · map (subtract 1) · trail n · uncounting
  trail n s = s ++ [n − (sum s + length s)]
```

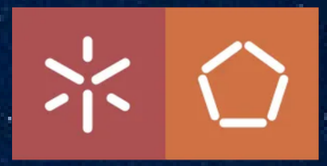
RLE decomposed

$$length \cdot (flip\ replicate\ a) = id$$

$$intercalate\ a\ (splitOn\ a\ x) = x$$

$$(subtract\ 1) \cdot (+1) = id$$

$$init\ (trail\ n\ xs) = xs$$



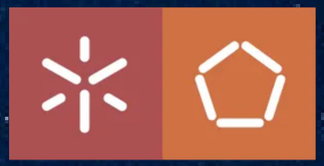
RLE decomposed

RLE encoding:

```
rle :: [Bit] → [Bit]
rle = counting · init · map (+1) · map length · splitOn [1]
```

RLE decoding:

```
unrle :: [Bit] → Int → [Bit]
unrle = flip pipeline where
  pipeline n = intercalate [1] · map (flip replicate 0) · map (subtract 1) · trail n · uncounting
  trail n s = s ++ [n − (sum s + length s)]
```



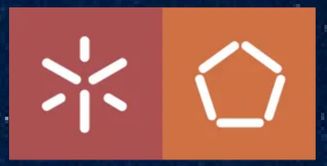
RLE decomposed

RLE encoding:

```
rle :: [Bit] → [Bit]
rle = counting · init · map (+1) · map length · splitOn [1]
```

RLE decoding:

```
unrle :: [Bit] → Int → [Bit]
unrle = flip pipeline where
  pipeline n = intercalate [1] · map (flip replicate 0) · map (subtract 1) · trail n · uncounting
  trail n s = s ++ [n − (sum s + length s)]
```



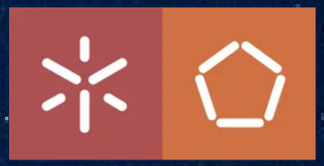
RLE decomposed

RLE encoding:

```
rle :: [Bit] → [Bit]
rle = counting · init · map (+1) · map length · splitOn [1]
```

RLE decoding:

```
unrle :: [Bit] → Int → [Bit]
unrle = flip pipeline where
  pipeline n = intercalate [1] · map (flip replicate 0) · map (subtract 1) · trail n · uncounting
  trail n s = s ++ [n − (sum s + length s)]
```



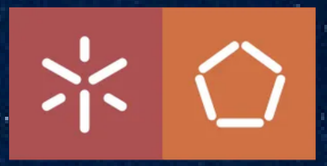
RLE decomposed

RLE encoding:

```
rle :: [Bit] → [Bit]
rle = counting · init · map (+1) · map length · splitOn [1]
```

RLE decoding:

```
unrle :: [Bit] → Int → [Bit]
unrle = flip pipeline where
  pipeline n = intercalate [1] · map (flip replicate 0) · map (subtract 1) · trail n · uncounting
  trail n s = s ++ [n − (sum s + length s)]
```



RLE decomposed

RLE encoding:

```
rle :: [Bit] → [Bit]
rle = counting · init · map (+1) · map length · splitOn [1]
```

RLE decoding:

```
unrle :: [Bit] → Int → [Bit]
unrle = flip pipeline where
  pipeline n = intercalate [1] · map (flip replicate 0) · map (subtract 1) · trail n · uncounting
  trail n s = s ++ [n − (sum s + length s)]
```

Counting

Calculation of *uncounting* Inverting *counting* to obtain *Uncounting* (a relation, hopefully a function in the end):

$$
\begin{aligned}
& Uncounting = counting^{\circ} \\
\equiv & \quad \{ \text{ definition of } counting \} \\
& Uncounting = (\!| cGen |\!)^{\circ} \\
\equiv & \quad \{ (46), (29), (28) \} \\
& Uncounting = [(cGen^{\circ})] \\
\equiv & \quad \{ \text{ let } Parse = cGen^{\circ} \text{ — a non-deterministic parser to begin with } \} \\
& Uncounting = [(Parse)]
\end{aligned}
$$

Parse

Unfolding *Parse*:

$$
\begin{aligned}
& Parse = cGen^{\circ} \\
\equiv & \quad \{ \text{ definition of } cGen; (40); (34); (29) \times 2 \ \} \\
& Parse = i_1 \cdot l^{\circ} \cup i_2 \cdot r^{\circ} \\
\equiv & \quad \{ \text{ definition of } cGen; (29) \times 2; (39) \ \} \\
& Parse = i_1 \cdot nil^{\circ} \cdot inj_{10}^{\circ} \cup i_2 \cdot (count^{\circ} \times id) \cdot conc^{\circ}
\end{aligned}
$$

Count

Inverting *count*:

$$
\begin{aligned}
& count = [in j_0 \cdot nil, g_2] \cdot outNat \cdot pred \\
\equiv & \quad \{\ (29)\ \times 2\ \} \\
& count^\circ = succ \cdot inNat \cdot [in j_0 \cdot nil, g_2]^\circ \\
\equiv & \quad \{\ inNat = [\underline{0}, succ];\ 41\ \} \\
& count^\circ = [\underline{1}, (2+)] \cdot [in j_0 \cdot nil, g_2]^\circ \\
\equiv & \quad \{\ 42\ \} \\
& count^\circ = \underline{1} \cdot (in j_0 \cdot nil)^\circ \cup (2+) \cdot g_2^\circ
\end{aligned}
$$

Milestone 3

Results and Discussion

Implementation status

Flags	Implemented
Mask Update	Implemented
RLE	Implemented + Verified
Resynchronization	Not implemented

Our contribution

Richer **formal specification** of the protocol

Haskell **implementation** with focus on the mathematical foundations as **new reference**

Correctness **verification** with relational algebra

Milestone 3

Conclusion and Future Work

Future work

Experiments are conducted on four corpora of housekeeping telemetry data. These were defined by the CCSDS Data Compression working group as a part of the standardization process of CCSDS 124.0-B-1.

- The *IceCube* corpus contains samples generated by a NASA cubesat using a commercial cloud radiometer for ice detection [28].
- The *Jason2* corpus consists of data produced by a NASA/CNES earth-observation satellite used for sea surface height measurements [29].
- The *PUS* corpus comprises synthetic packets simulated by the SimuPUS software developed by CNES to experiment new technologies.
- The *Venus* corpus contains samples from the Venus Express mission of ESA [30].

M. Hernández-Cabronero, D. Evans, J. Bartrina-Rapesta, F. Aulí-Llinàs, I. Blanes and J. Serra-Sagristà, "Resiliency and Efficiency of the CCSDS 124.0-B-1 Telemetry Compression Standard," in IEEE Access, vol. 12, pp. 36702-36711, 2024, doi: 10.1109/ACCESS.2024.3374227

Future work

Completing **proofs**

Exploring the **roles of flags** in properties

Implementation of **resynchronization** to verify actual packet loss properties

Conclusion

Foundation for a complete **proof of correctness** for the Pocket+ protocol.

Encourage further research into the formal verification of similar algorithms with critical applications, where correctness is of utmost importance, using **relational algebra**.

Thank you!