fn sample_uniform_ubig_below

Michael Shoemate

October 10, 2025

This document proves that the implementation of sample_uniform_ubig_below in mod.rs at commit f5bb719 (outdated[1]) satisfies its definition. This algorithm uses the same algorithm and argument as used for unsigned native integers, but this time the bit depth is dynamically chosen to fill the last byte of a series of bytes long enough to hold upper.

1 Hoare Triple

Preconditions

Compiler-verified

Argument upper is of type UBig, a bignum

Caller-verified

upper $\neq 0$

Pseudocode

```
def sample_uniform_ubig_below(upper: UBig) -> UBig:
    byte_len = upper.bit_len().div_ceil(8)
    max = Ubig.from_be_bytes([u8.MAX] * byte_len)
    threshold = max - max % upper

    buffer = [0] * byte_len
    while True:
        fill_bytes(buffer)

        sample = UBig.from_be_bytes(buffer)
        if sample < threshold:
            return sample % upper
```

Postcondition

sample_uniform_ubig_below either

- raises an exception if there is a lack of system entropy,
- returns out where out is uniformly distributed between $[0, upper)$.

[1] See new changes with git diff f5bb719..a8642172 rust/src/traits/samplers/uniform/mod.rs

2 Proof

Proof. `byte_len` is the fewest number of bytes necessary to represent `upper`, which works out to ceil(ceil($\log_2$(`upper`))/8). Let `max` denote the largest integer representable in this many bytes ($2^{(\texttt{byte\_len}\cdot 8)}-1$). Let `sample` be a uniform sample between $[0, \texttt{max}]$ by flipping `byte_len` $\cdot\, 8$ even coins.

You could then (naively) update the range of `sample` to $[0, \texttt{upper})$ by rejecting any `sample` greater than or equal to `upper`. To reduce the probability of rejection (and improve computational performance), partition the numbers into two sets:

- the leading `upper` $\cdot\, k =$ `threshold` numbers that wrap evenly modulo `upper`
- the remaining trailing (`max` mod `upper`) numbers

It is equivalent to only reject trailing numbers, and return the sample modulo `upper`. Since `max` = `threshold` + (`max` mod `upper`), then `threshold` = `max` − (`max` mod `upper`).
Therefore, for any value of `upper`, the function satisfies the postcondition. □

For an intuitive understanding of the sampling approach, consider when `upper` is two. The naive approach would sample one byte, and then reject if the integer is not zero or one, with probability $\frac{2^8-2}{2^8} \approx .99$. Alternatively, two valid outcomes wrap 2^7 times into 2^8, with no trailing numbers. Never reject `sample` because there are no trailing numbers, and return `sample` mod 2.

Now consider when `upper` is three. Three valid outcomes wrap 85 times into 2^8, with one trailing number (255). Reject if `sample` is 255, otherwise return `sample` mod 3.