Code suggestion for Query3 without modulo

Sebastian Heinekamp

May 4, 2021

Abstract

sketch of a algorithm that doesn't need to recreate the whole Int (for most cases) in order to decide solve the Modulo in Query3. Should allow to work with weave format more efficiently

1 Reformulate "Where-Clause"

$$a \mod b = c \Leftrightarrow \tag{1}$$

$$\exists \lambda \in \mathbb{N} : (a - c) - \lambda b = 0 \Leftrightarrow \tag{2}$$

$$\exists \lambda \in \mathbb{N} : \frac{(a - c)}{b} = \lambda \tag{3}$$

It should be possible exclude many rows without loading the complete numbers a,c and b, by checking for natural λ.

2 Pseudocode Propositon

Algorithm 1: check for natural λ bitwise

```
Result: if a natural λ exists such that it is the quotient of two numbers
initialization;
g := a - c;
if g < b by checking the first 1 in the bit strings then
    return false ;
else
    i = 1;
    j = 1;
    going through the bits from least to most significant
    while g[0:-j] > b[0:-i] by keeping track of most significant 1 do
        if length(g[0:-j]) == 0 then
            equals -j reaching the most significant position.
            return true;
        else
            if b[-i] == 0 then
                if g[-j] ==0 then
                    drop the bit -i and -j equals divide by two
                    j++; i++;
                    continue;
                else
                    g can't be divisible by two, while b is
                    return false;
                end
            else
                if g[-j] == 0 then
                    divide g by two
                    j++; continue;
                else
                    NON-Bitwise Block
                    subtract the remaining part of b from the remaining g
                      once
                    g[0:-j] -= b[0:-i];
                    continue;
                end
            end
        end
    end
end
```

It should be possible to figure out a way to keep track of the non bitwise block, such that one can correct the "new" bits. The rest of the code is index calculation or adresses only the last bit.

The runtime depends on the number of non-zero entries in g and the position of the least significant 1 in b. Worst case $\mathcal{O}(2 * size_t)$. The expected runtime depend on the distribution of a and c. many small c and large a are close to the worst case