Linear Natural Number Sequences

by Sven Nilsen, 2017

Here I describe some properties of linear natural number sequences. This demonstrates the practicality of path semantical notation to formalize mathematical objects using functions.

Path semantics makes it possible to write `x: [even] true` to express that `x` is an even number.

even(x) = (x % 2) == 0

The `even` function is an existential path to another algorithm:

even_sequence(x) = 2 * x

∃even_sequence <=> even

The `odd` function for natural numbers is also an existential path to another algorithm:

odd(x) = (x % 2) == 1

odd_sequence(x) = 1 + 2 * x

∃odd_sequence <=> odd

The `even_sequence` function has an inverse that is defined only for even numbers that are not equal to the first even number:

even_sequence_inv(x: [even] true) = x / 2

The `odd_sequence` function has an inverse that is defined only for odd numbers that are not equal to the first odd number:

odd_sequence_inv(x: [even] false) = (x – 1) / 2

All linear sequences of the natural numbers are of this form:

sequence(a, b: (> 0)) = \(x) = a + b * x

sequence(0, 2) <=> even_sequence
sequence(1, 2) <=> odd_sequence
sequence(0, 1) <=> inc

Linear sequences have an existential path that determines whether a number is in the sequence:

linear(a, b: (> 0)) = \(x) = if x < a { false } else { ((x – a) % b) == 0 }

∃sequence(a, b) <=> linear(a, b)

There is a partial inverse function that maps back to the set of all natural numbers:

sequence_inv(a, b: (> 0)) = \(x: [linear(a, b)] true) = (x – a) / b

The next and previous number can be found using partial functions:

next(a, b: (> 0)) = \(x: [linear(a, b)] true) = x + b

previous(a, b: (> 0)) = \(x: [linear(a, b)] true ∧ (¬= a)) = x – b

A linear number sequence can be a subset of another in particular way called `subset_offset`:

(x: [linear(a + b * n, b)] true) → (x: [linear(a, b)] true)

subset_offset((a_0, b_0), (a_1, b_1)) = (b_0 == b_1) && (a_0 >= a_1) && (((a_0 – a_1) % b_1) == 0)

strict_subset_offset((a_0, b_0), (a_1, b_1)) = (b_0 == b_1) && (a_0 > a_1) && (((a_0 – a_1) % b_1) == 0)

Another way a linear number sequence can be a subset of another is called `subset_step`:

(x: [linear(a, b * (> 0))] true) → (x: [linear(a, b)] true)

subset_step((a_0, b_0), (a_1, b_1)) = (a_0 == a_1) && (b_0 >= b_1) && ((b_0 % b_1) == 0)

strict_subset_step((a_0, b_0), (a_1, b_1)) = (a_0 == a_1) && (b_0 > b_1) && ((b_0 % b_1) == 0)

A third way a linear number sequence can be a subset of another is called simply `subset` or using the symbol `⊆`, since it handles both offset and step difference:

(x: [linear(a + b * n, b * (> 0))] true) → (x: [linear(a, b)] true)

(`⊆`) = \((a_0, b_0), (a_1, b_1)) =
(a_0 >= a_1) && (b_0 >= b_1) && (((a_0 – a_1) % b_1) == 0) && ((b_0 % b_1) == 0)

(`⊂`) = \((a_0, b_0), (a_1, b_1)) =
((a_0 > a_1) && (b_0 >= b_1) || (a_0 >= a_1) && (b_0 > b_1)) &&
(((a_0 – a_1) % b_1) == 0) && ((b_0 % b_1) == 0)

The strict subset function using the symbol `⊂` is a bit more complicated, because it has more edge cases.

You can create a pair that returns the sequence of the composition of two other number sequences. This means that you take a natural number, give it to one sequence, get a new natural number, give it to another sequence. Repeating for each number results in a new number sequence.

$$compose((a_0, b_0), (a_1, b_1)) = (a_1 + b_1 * a_0, b_1 * b_0)$$

Proof of the composition formula:

$$a_0 + b_0 * n_0$$
$$a_1 + b_1 * n_1$$
$$a_1 + b_1 * (a_0 + b_0 * n_0)$$
$$a_1 + b_1 * a_0 + b_1 * b_0 * n_0$$
$$(a_1 + b_1 * a_0) + (b_1 * b_0) * n0$$

By composing two linear number sequences, you construct a subset of the last sequence:

$$A := (A_a, A_b)$$
$$B := (B_a, B_b)$$
$$C := compose(A, B)$$

$$C \subseteq B$$

To compute the offset and step numbers that separates a sequence from its subset:

$$subset\_info((a_0, b_0), (a_1, b_1): (\subseteq (a_0, b_0))) = (a_1 - a_0, b_1 / b_0)$$

$((> 0), 1)$	The subset differs by offset
$(0, (> 1))$	The subset differs by step
$((> 0), (> 1))$	The subset differs by both offset and step

Notice that the subset info is itself a linear sequence, which is exactly the sequence that need to be composed with input to produce the subset.

$$compose(subset\_info(A, C)) = C$$

To detect whether two linear number sequences have a shared subset:

$$have\_shared\_subset((a_0, b_0), (a_1, b_1)) = (b_0 \neg= b_1) \,||\, ((a_0 \,\%\, b_0) == (a_1 \,\%\, b_1))$$

This can be intuitively understood when the two sequences have different steps, the shared subset will at least contain the product of two steps. For example, if one sequence has step 3 and the other 4, then the shared subset contains a sequence with step $3 * 4 = 12$.

On the hand, if they have the same step but differs by offset, then they only share a subset if their offsets syncs to the step of the other.

Obviously, if one sequence A has a subset B, then the largest shared subset is B.

For every step value, there are some lists of length equal to the step that contains the maximum number of sequences that have no shared subset in common. For example, for step 3:

(0, 3)	Starts at offset `0`
(1, 3)	
(2, 3)	
(3, 3)	Starts at offset `3`
(4, 3)	
(5, 3)	
(0, 3)	Varies offset
(4, 3)	
(2, 3)	

Picking the largest sequence from each list gives you `(0, 3), (1, 3), (2, 3)`. This is a unique list for every step.