Beurling Integers: Part 3

The Beurling Tree

Devin Platt

July 11, 2015

In the previous two articles we introduced the Beurling integers and discussed three isomorphic representations of them. In this post we discuss the partial order implied by the representation as a sequence of prime factorizations.

Recall our motivation. The distribution of the prime numbers is an important topic in mathematics with remaining mysteries. The Beurling generalized primes allow us to design "systems" of integers with any distribution of primes we choose, while maintaining many of the properties of regular integer multiplication, though notably removing its additive structure (we like to use the term "system" because it is representation-agnostic).

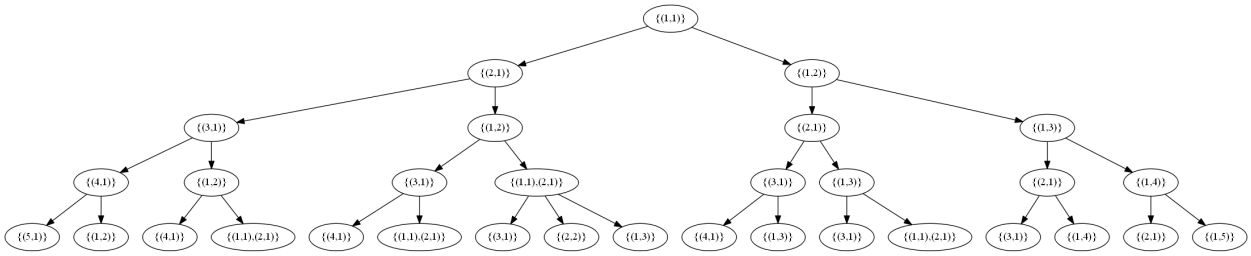
The idea of the "distribution of primes" is actually boring in the context of what is allowable for these generalized integer systems. The number 1 is neither prime nor composite, and the second number is always the first prime. Otherwise, anything goes. We can choose any subset of $\mathbb{N} \setminus \{1, 2\}$ and that will indicate the rest of our primes. If we wish, we might impose that this subset be infinite and that that its complement be infinite (so that there are infinitely many of both primes and composites), but this is limiting behavior so still we are left with whatever choice of primes we want up to any finite point. We also are still left with a great host of choices for asymptotic densities of primes.

The literature typically approaches this problem by asking what properties imply a certain asymptotic distribution of primes. Arne Beurling's original paper proved a version of the prime number theorem given a condition on the growth of the sequence of real integers. Other researchers have looked for properties like "even-spacing" – the Delone Property (see "Beurling Generalized Integers with the Delone Property" by Lagarias: http://www.math.lsa.umich.edu/ lagarias/doc/beurling.pdf).

We take a different approach here. We wont impose a new property and observe asymptotic growth rates of the primes. Rather, we will observe not just the distribution of primes, but the distribution of primes and composites, and we will do this to some finite point, not looking for asymptotic estimates. This is what will make our approach combinatorial instead of analytical. In the previous post we discussed prime factorizations as mathematical objects, and remarked that the Beurling integers form a partial order over prime factorizations. The rest of this series of posts will be devoted to investigating this partial order. Actually, there are two partial orders, because we also have a partial order on $\mathbb{N}^2$ (prime powers), but this second problem can be approached with the same techniques as the first.

Any partial order can be represented as a tree, and so our focus will be to build this tree and use it to study the order. This allows us to essentially to store all information about the Beurling generalized integers into one mathematical object, though we are limited by the fact that we can only actually build the tree to a finite height.

Figure 1: The Beurling Tree



Our first task will be to develop an algorithm to build the tree. Because the task is nontrivial, I want to give a higher-level view of it at first. We decompose the task into tree-building and child-decision problems. At first well view the child-decision subtask as a black box, or oracle.

1 Tree-Building

Each sequence begins with the identity followed by the first prime, so by convention we make the first prime the root of our tree. We start out with this root node r and decide to build our tree to some finite height h. Then we run the recursive routine AddChildren(r,h). Here is the algorithm for AddChildren:

Algorithm `AddChildren`(*node n, integer height*)
```
if height > 0 then
    add the next prime as a child of n;
    run the child decision algorithm on n to generate a set of factorizations;
    add the factorizations as children of n;
    foreach child c of n do
        AddChildren(c,height-1)
    end
end
```

Again, this is a high level view of the algorithm. In order to know what the next prime is we need to maintain a counter. We also need to keep information to run the child-decision subroutine. Please note that the child-decision routine returns a nonempty set. This means that each node has at least 2 children (a prime and at least one composite), and thus our tree grows exponentially with height.

2 Child-Decision

The main difficulty in our task will be that the order has complicated recursive rules – what is permissible for our choice of the next factorization in a sequence is dependent on not only what factorizations occur previously in the sequence (increasingness), but in what order they occur in (translation invariance). To make an efficient algorithm, we need a way to intelligently store this information.

Our solution is to use a multiplication table. The table will represent multiplication over our current sequence of factorizations as we build the tree. New entries will be pushed and popped off this table as we descend on and return from child nodes. The table will be triangular in the sense that multiplication is commutative and storing the same result twice would be redundant. We will

use the convention of storing $m(x, y)$ values where $x \leq y$. Increasingness will be obeyed in that a cells values cannot be filled until the cells above and to its left have been filled.

Translation invariance is a little trickier to maintain, but still possible with the table. At any point in the algorithm there will be a set of "frontier" cells, empty cells adjacent to filled in cells. These frontier cells are candidates for being next in our sequence because they would obey increasingness. Translation invariance says that if we have factorizations x and y, and $x < y$, then for all z we have $xz < yz$. The issue is that there are multiple ways of achieving a factorization. For example, with the natural numbers we have both $2 \cdot 6 = 12$ and $3 \cdot 4 = 12$. If a given factorization f is present in a frontier cell, then f is not a valid choice for the next item in our sequence unless all possible ways of obtaining f are in frontier cells.

Otherwise, translation invariance implies that some other factorization f is "less than" f, and it is required to occur first.

Thats the gist of our algorithm. To make things explicit, well run through the algorithm for a number of steps, giving the states of our data structures.

Notes:

- In this example we are building the tree to a height of four.
- The current path of the tree is highlighted.
- Frontier cells are marked with "Frontier".
- Redundant cells of the form (r, c) where $r > c$ are marked with an "x".

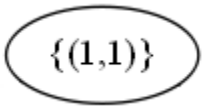
2.1 STEP 0 (initial)

:

Prime count: 1 Multiplication table:

	{(1,1)}

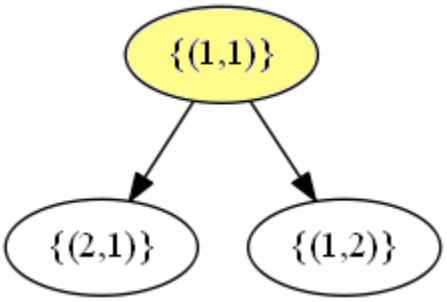
Figure 2: Step 0



2.2 Step 1

Prime count: 1 Multiplication table:

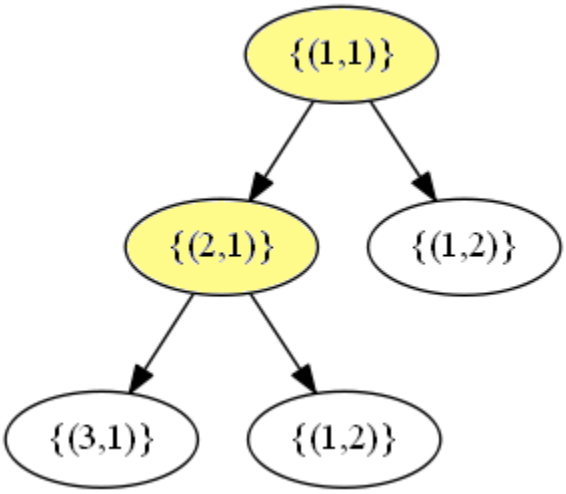
	{(1,1)}
{(1,1)}	Frontier



2.3 Step 2

Prime count: 2 Multiplication table:

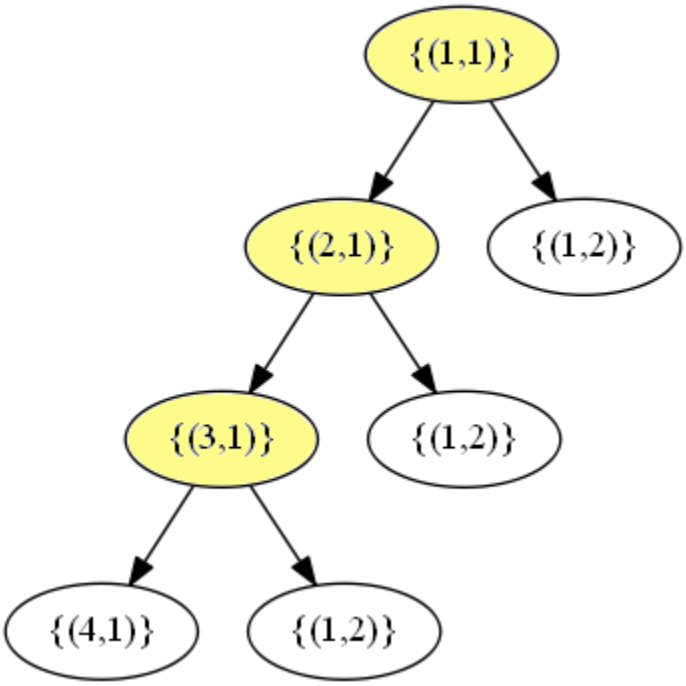
	{(1,1)}	{(2,1)}
{(1,1)}	Frontier	



2.4 Step 3

Prime count: 3 Multiplication table:

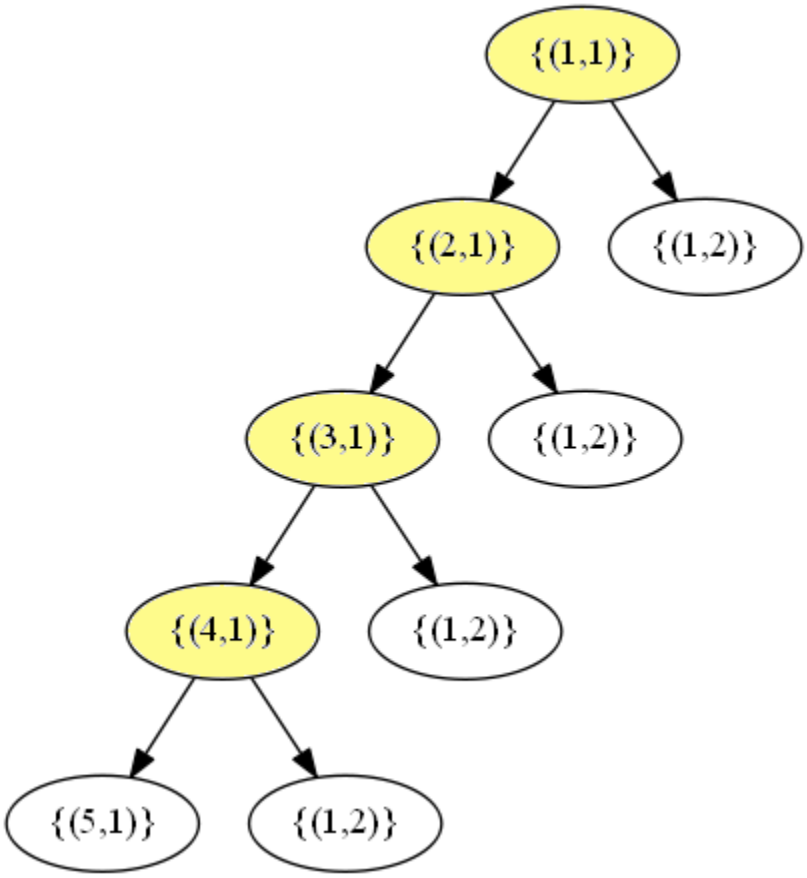
	{(1,1)}	{(2,1)}	{(3,1)}
{(1,1)}	Frontier		



2.5 Step 4

Prime count: 4 Multiplication table:

	{(1,1)}	{(2,1)}	{(3,1)}	{(4,1)}
{(1,1)}	Frontier			

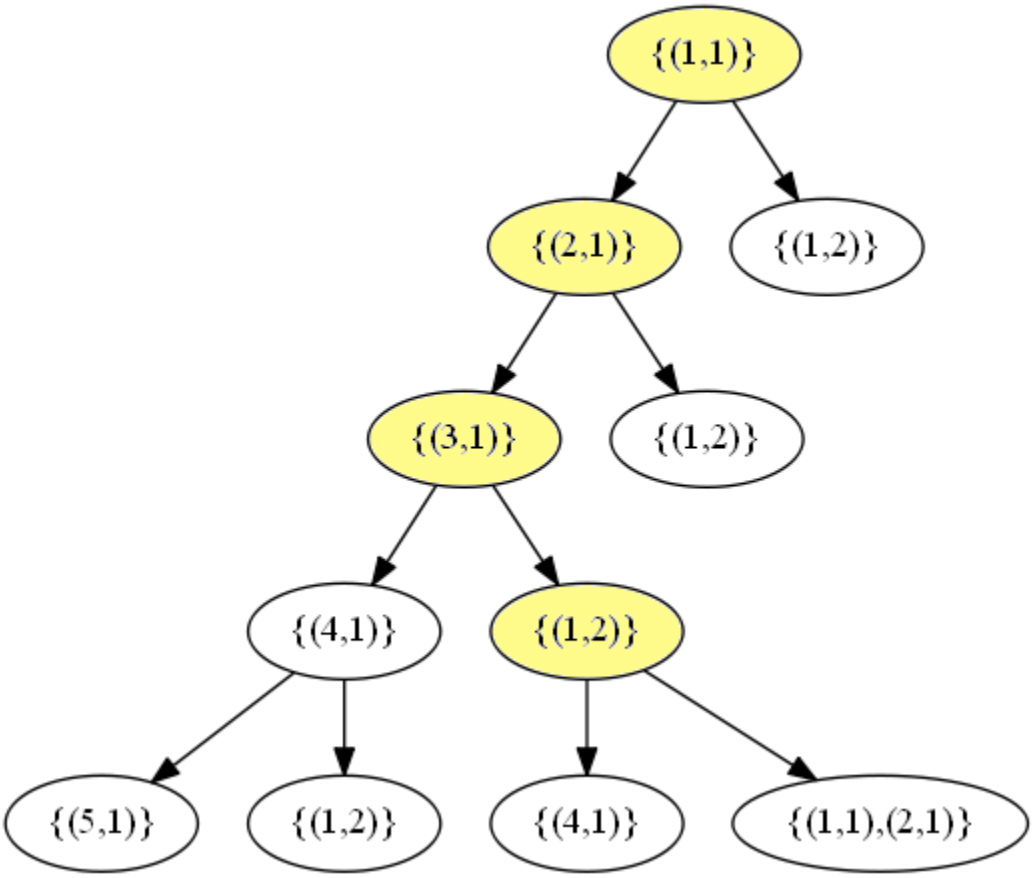


In the next step we'll back up as we have reached the maximum height.

2.6 Step 5

Prime count: 3 Multiplication table:

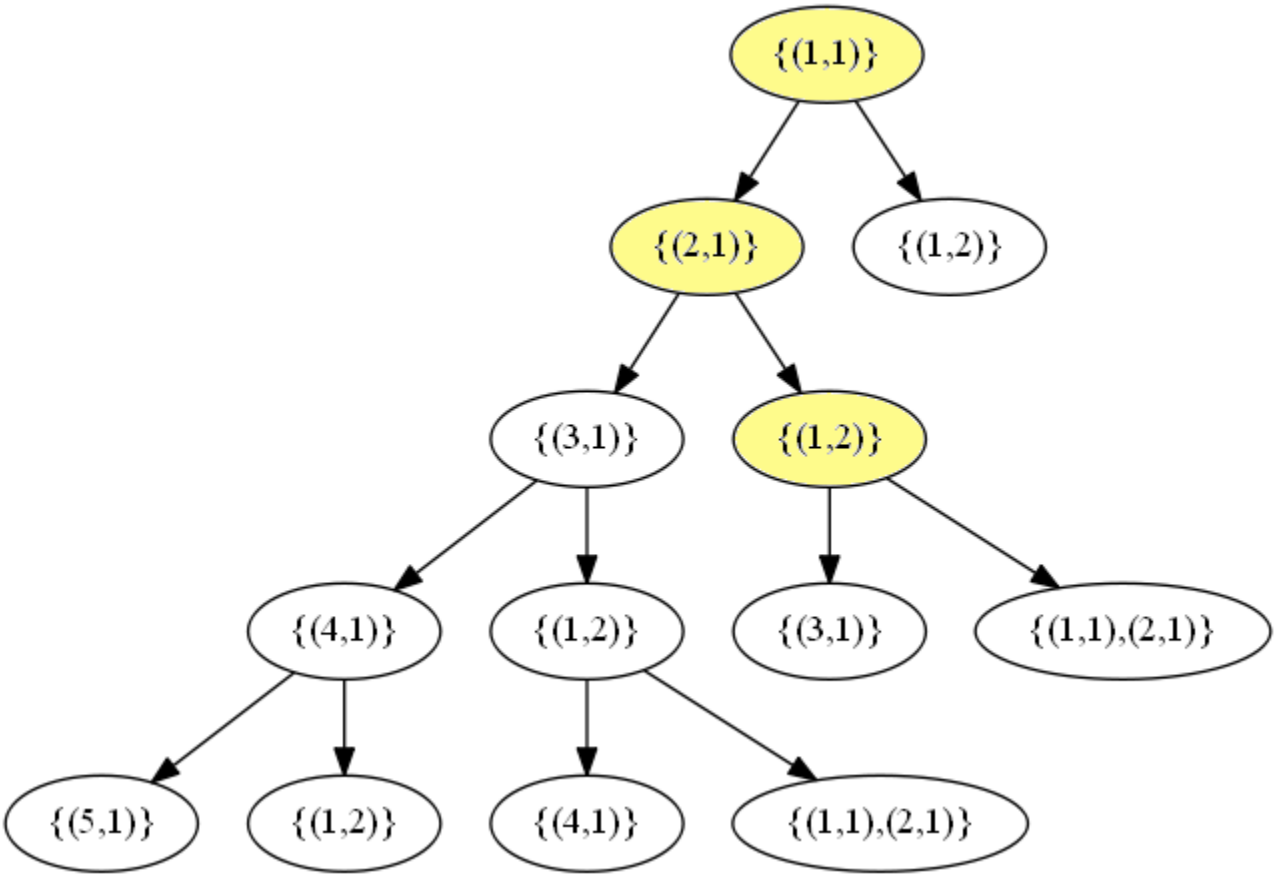
	{(1,1)}	{(2,1)}	{(3,1)}	{(1,2)}
{(1,1)}	{(1,2)}	Frontier		
{(2,1)}	x			



2.7 Step 6

Prime count: 2 Multiplication table:

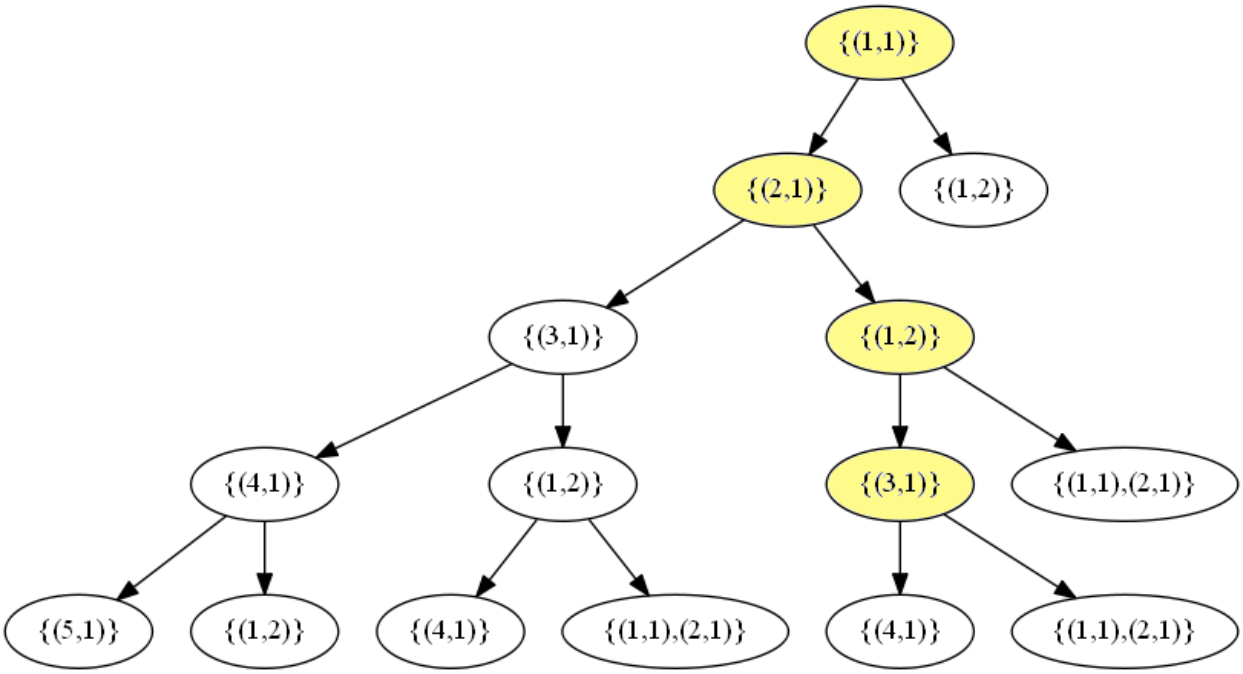
	{(1,1)}	{(2,1)}	{(1,2)}
{(1,1)}	{(1,2)}	Frontier	
{(2,1)}	x		



2.8 Step 7

Prime count: 3 Multiplication table:

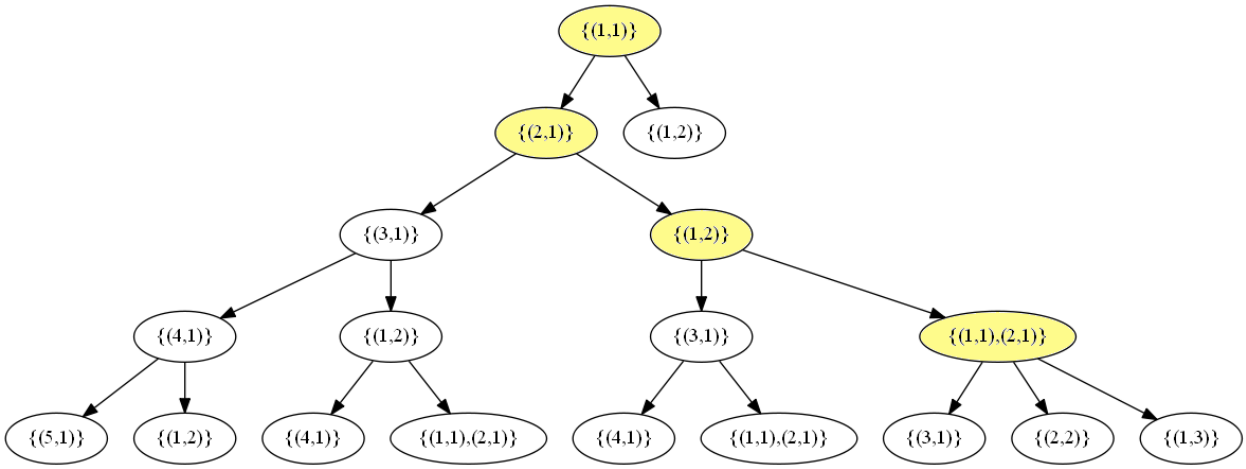
	{(1,1)}	{(2,1)}	{(1,2)}	{(3,1)}
{(1,1)}	{(1,2)}	Frontier		
{(2,1)}	x			



2.9 Step 8

Prime count: 2 Multiplication table:

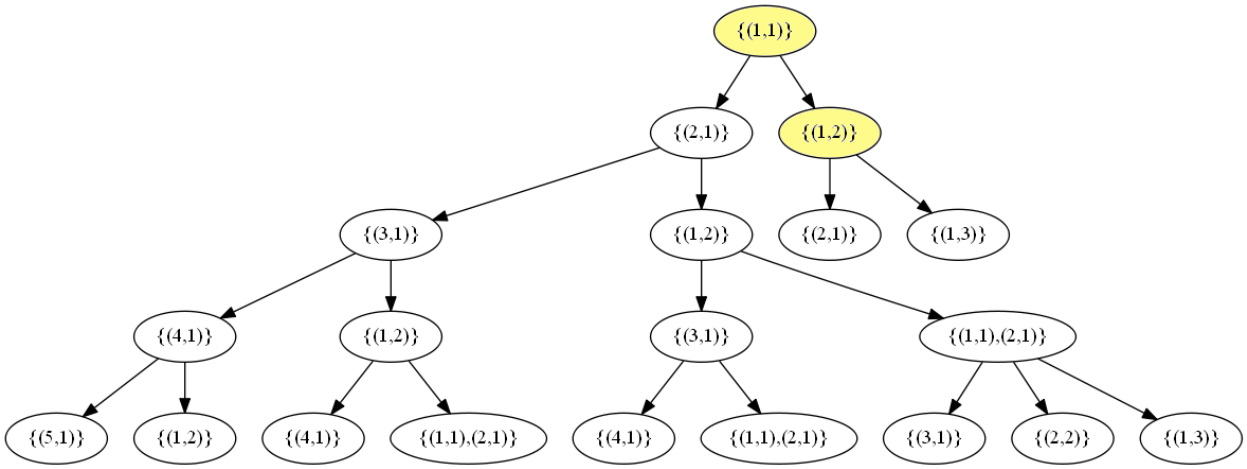
	{(1,1)}	{(2,1)}	{(1,2)}	{(1,1),(2,1)}
{(1,1)}	{(1,2)}	{(1,1),(2,1)}	Frontier	
{(2,1)}	x	Frontier		



2.10 Step 9

Prime count: 1 Multiplication table:

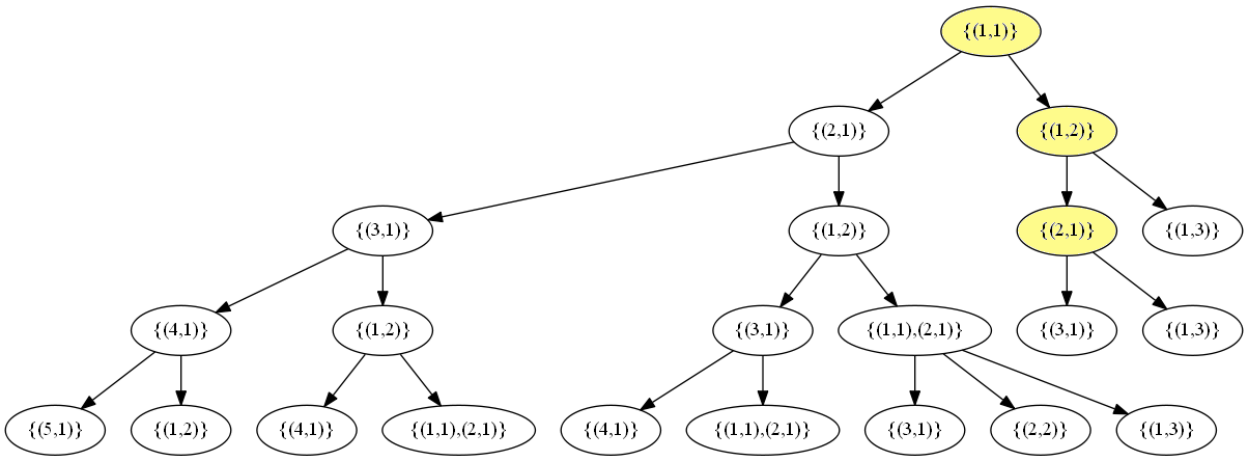
	{(1,1)}	{(1,2)}
{(1,1)}	{(1,2)}	Frontier
{(1,2)}	x	



2.11 Step 10

Prime count: 2 Multiplication table:

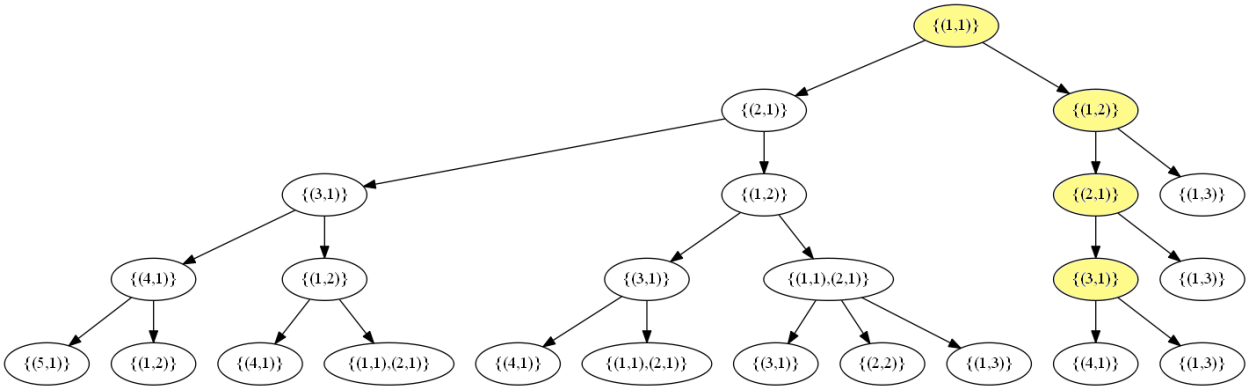
	{(1,1)}	{(1,2)}	{(2,1)}
{(1,1)}	{(1,2)}	Frontier	
{(1,2)}	x		



2.12 Step 11

Prime count: 3 Multiplication table:

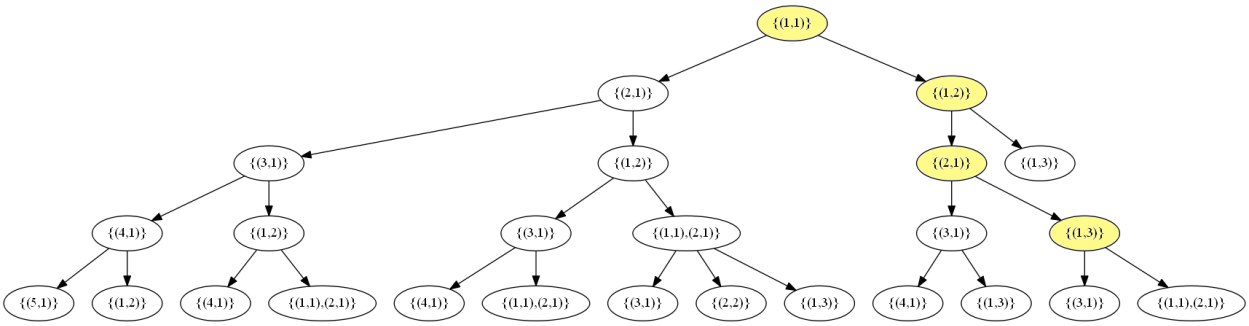
	{(1,1)}	{(1,2)}	{(2,1)}	{(3,1)}
{(1,1)}	{(1,2)}	Frontier		
{(1,2)}	x			



2.13 Step 12 (This is an example of rejecting a Frontier cell due to translation invariance)

Prime count: 2 Multiplication table:

	{(1,1)}	{(1,2)}	{(2,1)}	{(1,3)}
{(1,1)}	{(1,2)}	{(1,3)}	Frontier	
{(1,2)}	x	Frontier		

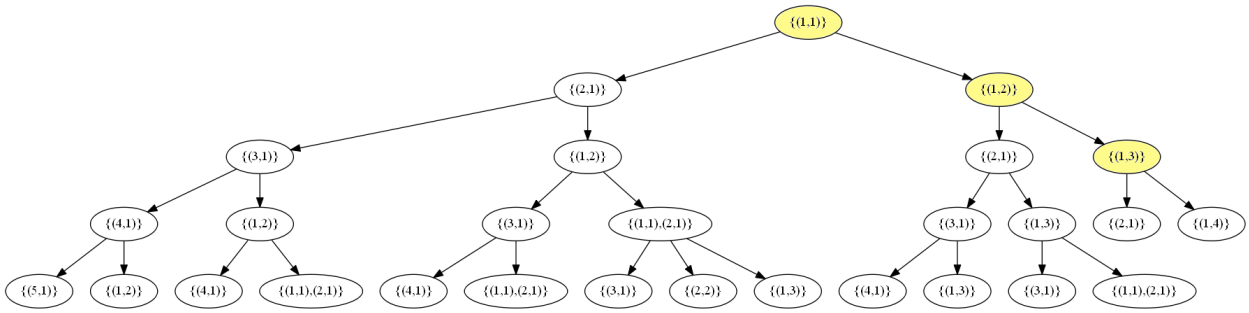


Note that the factorization $\{(1,4)\}$ requires two different frontier cells: the cell corresponding to $\{(1,1)\}$ and $\{(1,3)\}$, and the cell corresponding to $\{(2,2)\}$ and $\{(2,2)\}$. So although we have two different factorizations in Frontier cells, only one of them ($\{(1,1),(2,1)\}$) is valid. (Figuring out how many cells are necessary for a given factorization has a relatively simple algorithm, which is explained in my implementation of this tree building process.)

2.14 Step 13

Prime count: 1 Multiplication table:

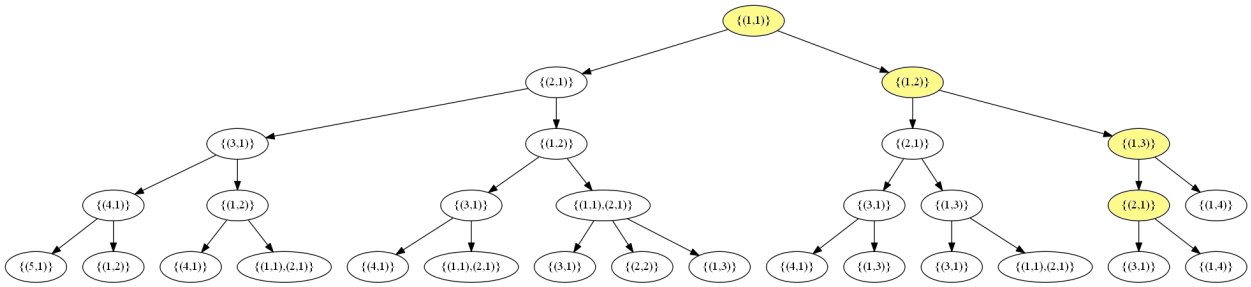
	{(1,1)}	{(1,2)}	{(1,3)}
{(1,1)}	{(1,2)}	{(1,3)}	Frontier
{(1,2)}	x	Frontier	



2.15 Step 14

Prime count: 2 Multiplication table:

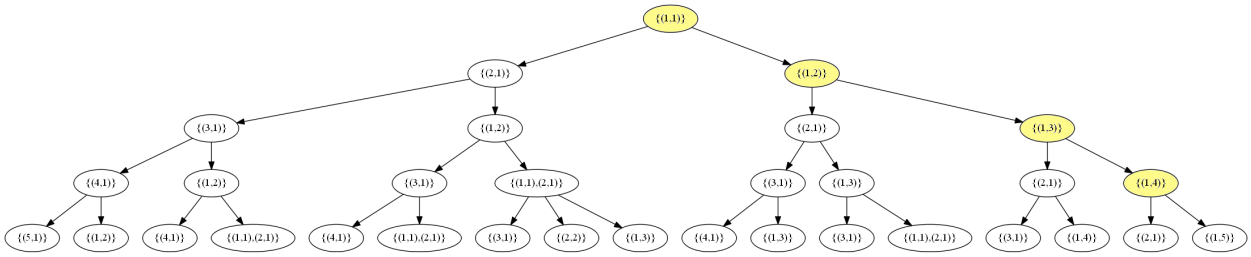
	{(1,1)}	{(1,2)}	{(1,3)}	{(2,1)}
{(1,1)}	{(1,2)}	{(1,3)}	Frontier	
{(1,2)}	x	Frontier		



2.16 Step 15

Prime count: 1 Multiplication table:

	{(1,1)}	{(1,2)}	{(1,3)}	{(1,4)}
{(1,1)}	{(1,2)}	{(1,3)}	{(1,4)}	Frontier
{(1,2)}	x	{(1,4)}	Frontier	
{(1,3)}	x	x		



An implementation of the algorithm is available on my Github (https://github.com/devinplatt/BeurlingTree). See the next article for analysis of the tree.