4.7 Proofs and Programming II: Prime Numbers

In the previous section, we saw how mathematical definitions and proofs can be used to create new implementations of functions, justifying the correctness of those implementations.

Now, we'll look at one additional example of this drawn from number theory: checking whether a number is prime. As we'll see later in this course, prime numbers are a fundamental building block of many techniques for encrypting data, and so being able to pick large prime numbers is a key part of many cryptographic algorithms.

Definition of prime numbers

As usual, we'll start with a formal definition of prime numbers.

Definition. Let $p \in \mathbb{Z}$. We say p is **prime** when it is greater than 1 and the only natural numbers that divide it are 1 and itself.

Let's define a predicate $IsPrime(p)$ to express the statement that "p is a prime number," with and without using the divisibility predicate.

The first part of the definition, "greater than 1", is straightforward. The second part is a bit trickier, but a good insight is that we can enforce constraints on values through implication: *if a natural number d divides p, then $d = 1$ or $d = p$*. We can put these two ideas together to create a definition in predicate logic:

$$IsPrime(p) : p > 1 \land \big(\forall d \in \mathbb{N},\ d \mid p \Rightarrow d = 1 \lor d = p\big), \qquad \text{where } p \in \mathbb{Z}$$

For practice, we can also unpack the definition of divisibility in this definition. We've underlined the changed part below.

$$IsPrime(p) : p > 1 \land \big(\forall d \in \mathbb{N},\ \underline{(\exists k \in \mathbb{Z},\ p = kd)} \Rightarrow d = 1 \lor d = p\big), \quad \text{where } p \in \mathbb{Z}$$

Translating $IsPrime$ to `is_prime`

Here's a start for translating the definition of prime into a Python function:

```python
def is_prime(p: int) -> bool:
    """Return whether p is prime."""
```

Just as we saw with `divides` in the previous section, we have a problem of translating quantification over an infinite set in the definition: $\forall d \in \mathbb{N}$. We can use the same property of divisibility as above and note that the possible natural numbers that are divisors of `p` are in the set $\{1, 2, \dots, p\}$. The implication $d \mid p \Rightarrow d = 1 \lor d = p$ is a bit harder to translate, so here we recall what we discussed in 3.3 Filtering Collections to use the `if` keyword in a comprehension to model implications. Here is our complete implementation of `is_prime`:

```python
def is_prime(p: int) -> bool:
    """Return whether p is prime."""
    possible_divisors = range(1, p + 1)
    return (
        p > 1 and
        all({d == 1 or d == p for d in possible_divisors i
    )
```

Notice that just like the mathematical definition, in Python our implementation of `is_prime` uses the `divides` function. This is a great example of how useful it can be to divide our programs into functions that build on each other, rather than writing all of our code in a single function. As we learn about more complex domains in this course, we'll see this pattern repeat itself: definitions will build on top of one another, and you should expect that your functions will build on one another as well.

Making our implementation faster

This implementation is a direct translation of the mathematical definition of prime numbers, with the only difference being our restriction of the range of possible divisors. However, you might have noticed that this algorithm is "inefficient" because it checks more numbers than necessary.

Often when this version of `is_prime` is taught in a programming context, the range of possible divisors extends only to the square root of the input `p`. Here is our second implementation of the function:

```python
from math import floor, sqrt


def is_prime_v2(p: int) -> bool:
    """Return whether p is prime."""
    possible_divisors = range(2, floor(sqrt(p)) + 1)
    return (
        p > 1 and
        all({not divides(d, p) for d in possible_divisors}
    )
```

This version is intuitively faster, as the range of possible divisors to check is smaller. But how do we actually know that this version of `is_prime` is correct? We could write some tests, but as we discussed earlier both unit tests and property-based tests do not guarantee absolute correctness, they just give confidence. Luckily, for algorithms like this one that are based on the mathematical properties of the input, we do have a tool that guarantees absolutely certainty: proofs!

A property of prime numbers

Formally, we can justify the correctnessof by formally proving the following statement.

Theorem. Let $p \in \mathbb{Z}$. Then p is prime if and only if $p > 1$ and for every integer d in the range $2 \leq d \leq \sqrt{p}$, d does not divide p.

Or, translated into predicate logic:

$$\forall p \in \mathbb{Z},\ IsPrime(p) \Leftrightarrow \big(p > 1 \land (\forall d \in \mathbb{N},\ 2 \leq d \leq \sqrt{p} \Rightarrow d \nmid p)\big).$$

How do we go about proving that this statement is correct? Because it is a biconditional, we can treat it as two implications, and prove each implication separately.

Proving the first implication

Discussion. The first implication we'll prove is that if p is prime, then $p > 1$ and $\forall d \in \mathbb{N},\ 2 \leq d \leq \sqrt{p} \Rightarrow d \nmid p$. We get to assume that p is prime, and will need to prove two things: that $p > 1$, and that $\forall d \in \mathbb{N},\ 2 \leq d \leq \sqrt{p} \Rightarrow d \nmid p$.

Let's remind ourselves what the definition of prime is in predicate logic:

$$Prime(p) :\ p > 1 \land \big(\forall d \in \mathbb{N},\ d \mid p \Rightarrow d = 1 \lor d = p\big)$$

The first part comes straight from the definition of prime. For the second part, we should also be able to use the definition of prime: if d is between 2 and $\sqrt{p}$, then it can't equal 1 or p, which are the only possible divisors of p.

Let's see how to write this up formally.

Proof. Let $p \in \mathbb{Z}$ and assume that p is prime. We need to prove that $p > 1$ and for all $d \in \mathbb{N}$, if $2 \leq d \leq \sqrt{p}$ then d does not divide p.

Part 1: proving that $p > 1$.

By the definition of prime, we know that $p > 1$.

Part 2: proving that for all $d \in \mathbb{N}$, if $2 \leq d \leq \sqrt{p}$ then d does not divide p.

Let $d \in \mathbb{N}$ and assume $2 \leq d \leq \sqrt{p}$. We'll prove that d does not divide p.

First, since $2 \leq d$, we know $d > 1$, and so $d \neq 1$. Second, since $p > 1$, we know that $\sqrt{p} < p$, and so $d \leq \sqrt{p} < p$.

This means that $d \neq 1$ and $d \neq p$. By the definition of prime again, we can conclude that $d \nmid p$. ■

What we've proved so far is that if p is prime, then it has no divisors between 2 and $\sqrt{p}$. How does this apply to `is_prime_v2`? When its input `p` is a prime number, we know that the expressions `p > 1` and `all(not divides(d, p) for d in possible_divisors)` will both evaluate to `True`, and so the function will return `True`. In other words, we've proven that `is_prime_v2` returns the correct value for *every* prime number, without a single test case! Pretty awesome.

Proving the second implication

Though we know that `is_prime_v2` is correct for prime numbers, we've said nothing at all about how it behaves when given a non-prime number. To prove that its behaviour is correct in this case as well, we need to prove the other implication.

Discussion. We now need to prove the second implication, which is the converse of the first: if $p > 1$ and $\forall d \in \mathbb{N},\ 2 \leq d \leq \sqrt{p} \Rightarrow d \nmid p$, then p must be prime. Expanding the definition of prime, we need to prove that $p > 1$ (which we've assumed!) and that for all $d_1 \in \mathbb{N},\ d_1 \mid p \Rightarrow d_1 = 1 \lor d_1 = p$.

So the idea here is to let $d_1 \in \mathbb{N}$ and assume $d_1 \mid p$, and use the condition that $\forall d \in \mathbb{N},\ 2 \leq d \leq \sqrt{p} \Rightarrow d \nmid p$ to prove that d_1 is 1 or p.

Proof. Let $p \in \mathbb{N}$, and assume $p > 1$ and that $\forall d \in \mathbb{N},\ 2 \leq d \leq \sqrt{p} \Rightarrow d \nmid p$. We want to prove that p is prime, i.e., that $p > 1$ and that $\forall d_1 \in \mathbb{N},\ d_1 \mid p \Rightarrow d_1 = 1 \lor d_1 = p$.

We know the first part ($p > 1$) is true because it's one of our assumptions. For the second part, first let $d_1 \in \mathbb{N}$, and assume $d_1 \mid p$. We'll prove that $d_1 = 1 \lor d_1 = p$.

From our second assumption, we know that since $d_1 \mid p$, it is not between 2 and $\sqrt{p}$.[1] So then either $d_1 < 2$ or $d_1 > \sqrt{p}$. We divide our proof into two **cases** based on these possibilities.

[1] More precisely, the *contrapositive* of our second assumption says that for all $d \in \mathbb{N}$, $d \mid p \Rightarrow d < 2 \lor d > \sqrt{p}$.

Case 1: assume $d_1 < 2$.

Since $d_1 \in \mathbb{N}$, it must be 0 or 1 in this case. We know $0 \nmid p$ because $p > 1$, and so $d_1 = 1$.

Case 2: assume $d_1 > \sqrt{p}$.

Since we assumed $d_1 \mid p$, we expand the definition of divisibility to conclude that $\exists k \in \mathbb{Z},\ p = d_1 k$. Since $d_1 > \sqrt{p}$ in this case, we know that $k = \frac{p}{d_1} < \frac{p}{\sqrt{p}} = \sqrt{p}$.

Since $p = d_1 k$, we know that $k \mid p$ as well, and so our second assumption applied to k tells us that k is not between 2 and $\sqrt{p}$.

So $k < \sqrt{p}$ and is not between 2 and $\sqrt{p}$. Therefore $k = 1$, and so $d_1 = \frac{p}{k} = p$. ■

To wrap up this example, let's see how this implication connects to our function `is_prime_v2`. What we've proved is that if `is_prime_v2(p)` returns `True`, then `p` must be prime. This sounds very similar to what we said in the previous section, but it is different! The contrapositive this statement here is useful: if `p` is NOT prime, then `is_prime_v2(p)` returns `False`.

So putting the two implications together, we have:

- For all integers `p`, if `p` is prime then `is_prime_v2(p)` returns `True`.
- For all integers `p`, if `is_prime_v2(p)` returns `True` then `p` is prime.[2]

[2] Or equivalently, if `p` is not prime then `is_prime_v2(p)` returns `False`.

Since every integer `p` is either prime or not prime, we can conclude that this implementation of `is_prime_v2` is **correct** according to its specification.