1 Lambda Calculus

1.1 Programming in Lambda Calculus

1.1.1 Recursion

Suppose we want to implement a function in Lambda Calculus that sums up the natural numbers up to n, e.g. $0 + 1 + 2 + \cdots + n$.

In Python, we might do something like:

```
def sum(n):
    if n == 0:
        return 0
    else
        return n + sum(n - 1)
```

In Lambda Calculus, we don't have named functions. The idea, then, is:

- We want to call a function on `DEC n`.
- And it needs to be the same function.

First, we need to pass in the function to call "recursively":

```
let STEP =
    \rec -> \n -> ITE (ISZ n)
                    ZERO
                    (ADD n (rec (DEC n))) -- Call some rec
```

We now need to do something clever to `STEP` so that the function is passed as `rec` itself. To do this, we want a **fixpoint combinator** `FIX` such that `FIX STEP` calls `STEP` with itself as the first argument; that is:

```
FIX STEP
=*> STEP (FIX STEP)
```

Once we have this, we can define

```
let SUM = FIX STEP
```

Then, by the property of `FIX`, we have

```
SUM *> STEP SUM -- (1)
```

To see this:

```
eval sum_one:
    SUM ONE
    =*> STEP SUM ONE
    =~> ONE
```