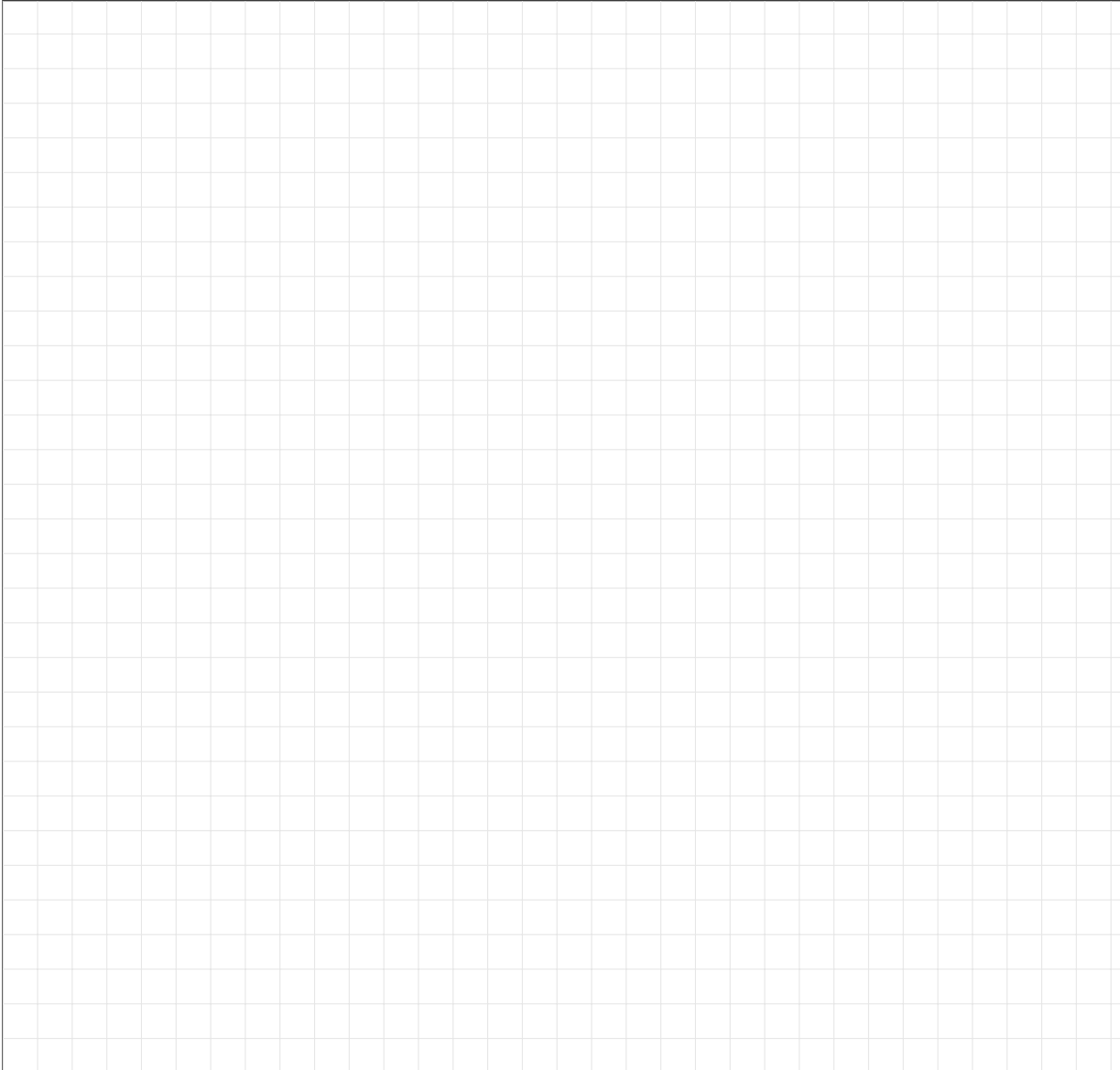


**Exercise 1 (Fixed-point Combinator)**

- a) In the last tutorial, we came up with an encoding for lists together with the functions `nil`, `cons`, `null`, `hd`, and `tl`. Use a fixed-point combinator to compute the length of a list in this encoding.
- b) In the last homework, we encoded lists with the fold encoding, i.e. a list  $[x, y, z]$  is represented as  $\lambda cn. cx(cy(czn))$ . Define a length function for lists in this encoding.



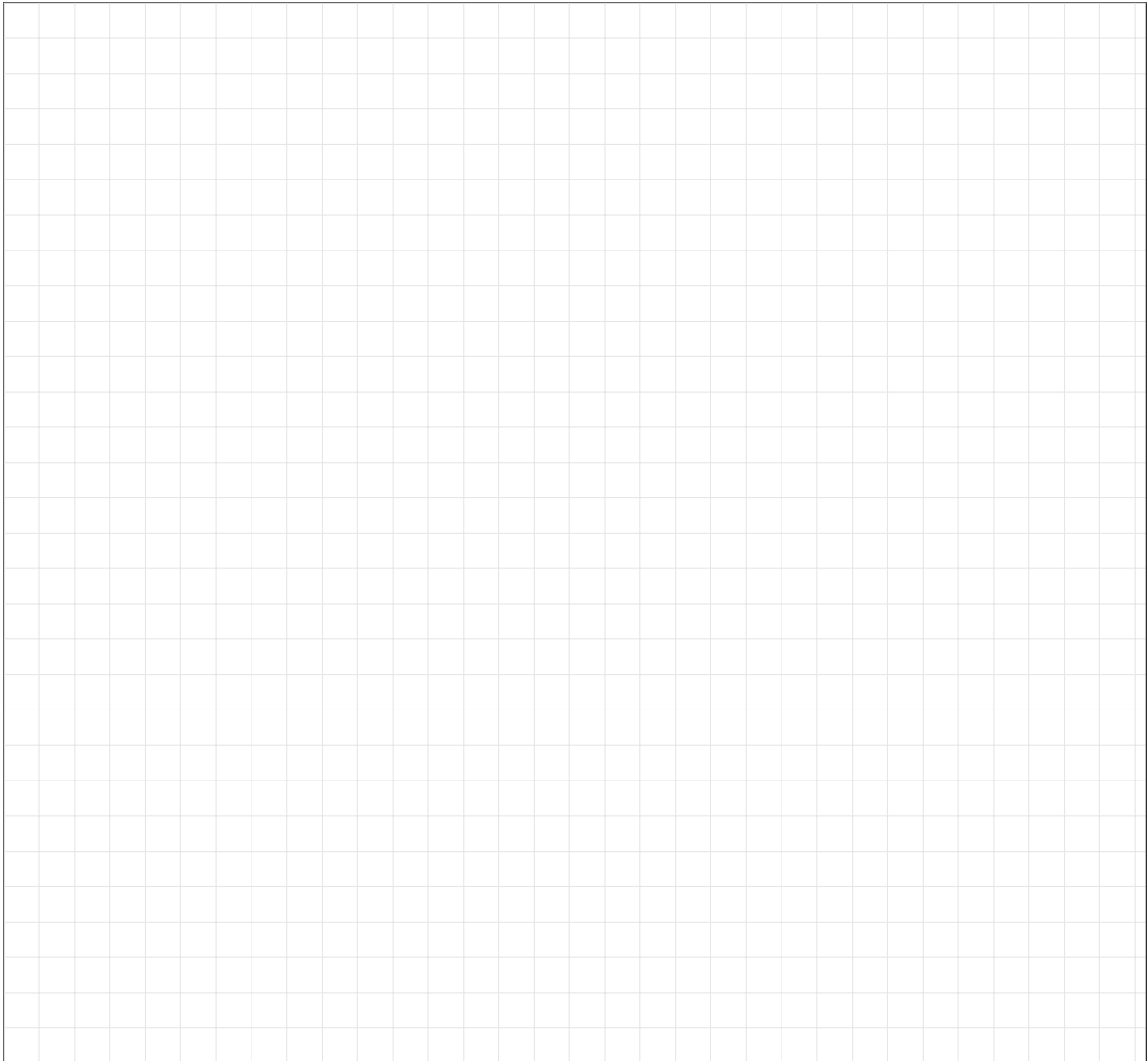
## Exercise 2 ( $\beta$ -reduction on de Bruijn Preserves Substitution)

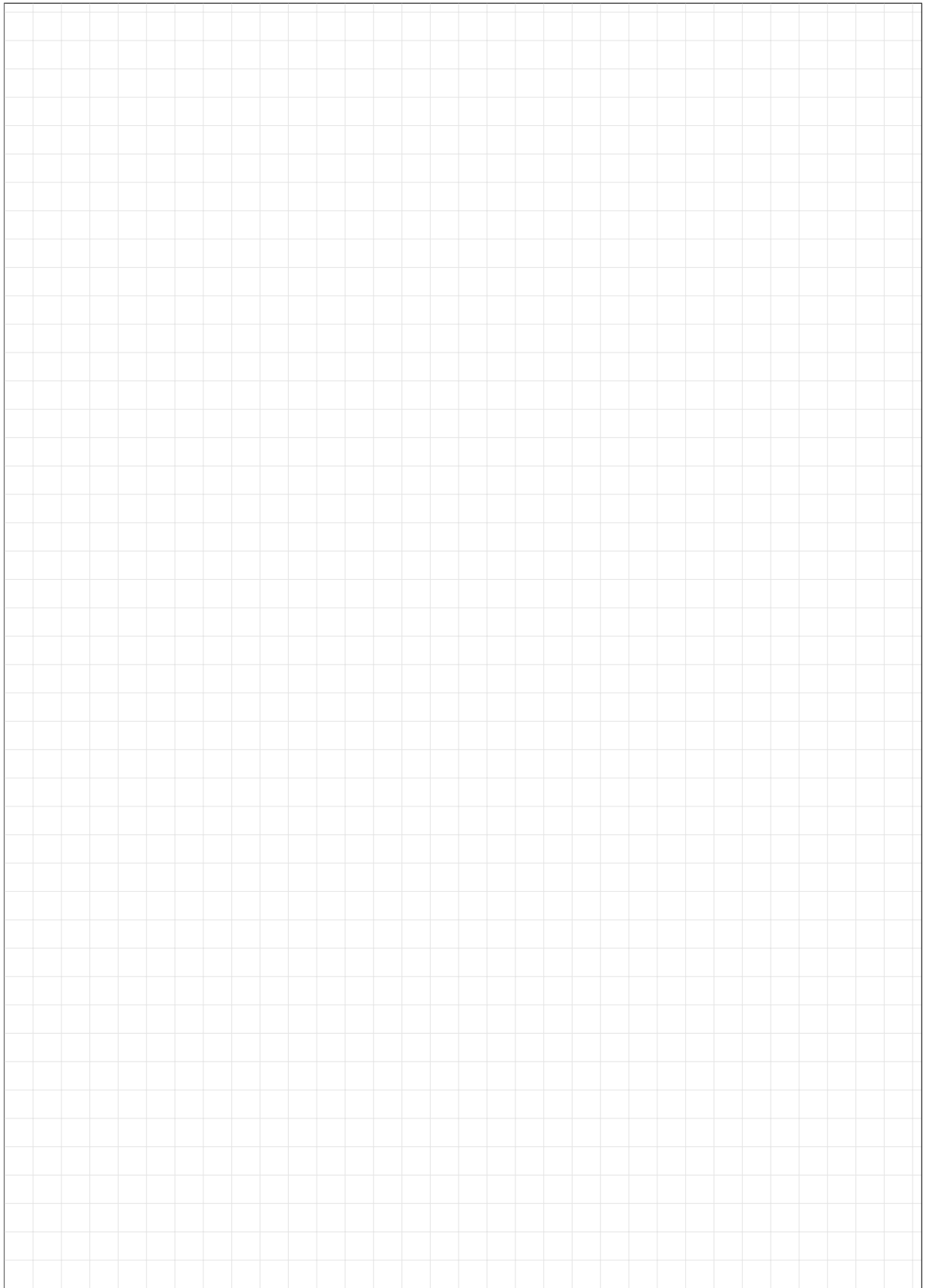
We consider an alternative representation of  $\lambda$ -terms that is due to de Bruijn. In this representation,  $\lambda$ -terms are defined according to the following grammar:

$$d ::= i \in \mathbb{N}_0 \mid d_1 d_2 \mid \lambda d$$

- Convert the terms  $\lambda x y. x$  and  $\lambda x y z. x z (y z)$  into terms according to de Bruijn.
- Convert the term  $\lambda ((\lambda (1 (\lambda 1))) (\lambda (2 1)))$  into our usual representation.
- Define substitution and  $\beta$ -reduction on de Bruijn terms.
- Now restate Lemma 1.2.5 for de Bruijn terms and prove it:

$$s \rightarrow_{\beta} s' \implies s[u/x] \rightarrow_{\beta} s'[u/x]$$





### Homework 3 (Multiplication)

Define multiplication using `fix` and prove its correctness. You can assume that you are given a predecessor function `pred` such that:

- $\text{pred } \underline{0} \rightarrow_{\beta}^* \underline{0}$
- $\text{pred } (\text{succ } n) \rightarrow_{\beta}^* n$

### Homework 4 (Efficient Substitution on de Bruijn)

We define a new lifting operator  $- \uparrow_{\square}$ :

$$i \uparrow_l^n = \begin{cases} i, & \text{if } i < l \\ i + n, & \text{if } i \geq l \end{cases}$$
$$(d_1 d_2) \uparrow_l^n = d_1 \uparrow_l^n d_2 \uparrow_l^n$$
$$(\lambda d) \uparrow_l^n = \lambda d \uparrow_{l+1}^n$$

Use  $- \uparrow_{\square}$  to define a more efficient version of substitution for de Bruijn terms that only applies lifting in the case that a variable is actually replaced by a term. Prove that  $t[s/0]$  yields the same result for both, your new version and the version from the tutorial. *Hint:* Find a suitable generalization first.

### Homework 5 (Expanding Lets)

We have a language with `let`-expressions, i.e.:

$$t ::= v \mid t t \mid \text{let } v = t \text{ in } t$$

Write a program which expands all `let`-expressions. The `let`-semantics are:

$$(\text{let } v = t_1 \text{ in } t_2) = (\lambda v. t_2) t_1$$

You can find a Haskell template for this exercise [here](#).