

# Implementing inductively defined translations of System $T$ terms in a graphical user interface

Kieran Maharaj

Master of Computing in Computer Science and Mathematics  
The University of Bath  
Year 3

This dissertation may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purposes of consultation.

# Implementing inductively defined translations of System $T$ terms in a graphical user interface

Submitted by: Kieran Maharaj

## Copyright

Attention is drawn to the fact that copyright of this dissertation rests with its author. The Intellectual Property Rights of the products produced as part of the project belong to the author unless otherwise specified below, in accordance with the University of Bath's policy on intellectual property (see [https://www.bath.ac.uk/publications/university-ordinances/attachments/Ordinances\\_1\\_October\\_2020.pdf](https://www.bath.ac.uk/publications/university-ordinances/attachments/Ordinances_1_October_2020.pdf)).

This copy of the dissertation has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the dissertation and no information derived from it may be published without the prior written consent of the author.

## Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

### **Abstract**

Terms in System  $T$ , a  $\lambda$ -calculus, are defined inductively, yet tools that perform operations on terms in any  $\lambda$ -calculus tend to force users to input them from left to right as text. The central purpose of this project is to implement an interface for inputting terms for inductive translations. Currently there are tools to inputting terms to perform translations, but they compel users to input terms as text rather than in an inductive way. A translation, known as 'modulus of continuity' has been implemented in Haskell. Blockly, a tool to construct visual programming languages, has been used to provide the user interface as it is well-suited to the task of inputting inductively defined terms. I was not able to find a tool that allowed users to input a term in a  $\lambda$ -calculus in an inductive way, using a block-based visual programming language, so this is filling a gap. It has been successful - the tool can be used to compute moduli of continuity using a block-based visual programming interface. I hope there is more development of block-based visual programming languages specifically designed for  $\lambda$ -calculus, because System  $T$  and Blockly are not completely suited to each other: more work should be done to investigate this further.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Literature and Technology Survey</b>	<b>2</b>
2.1	Theoretical underpinning . . . . .	2
2.1.1	The Ordinary System $T$ . . . . .	2
2.1.2	Comparison between the input language and the output language . . . . .	3
2.1.3	Inductive functions in System $T$ . . . . .	3
2.1.4	Continuity in System $T$ . . . . .	6
2.1.5	b-translation . . . . .	6
2.1.6	Examples . . . . .	7
2.2	Existing tools . . . . .	9
2.2.1	Agda implementation . . . . .	9
2.2.2	Mikrokosmos . . . . .	9
<b>3</b>	<b>Requirements</b>	<b>10</b>
3.1	Achieved requirements . . . . .	10
3.2	Discarding of certain requirements . . . . .	10
3.2.1	Division between intermediate computations and final computation . . . . .	10
3.2.2	Complexity . . . . .	10
3.2.3	Hosting . . . . .	10
3.3	Lists of requirements . . . . .	10
<b>4</b>	<b>Design</b>	<b>13</b>
4.1	Initial thoughts . . . . .	13
4.1.1	Interface . . . . .	13
4.1.2	Interaction . . . . .	13
4.1.3	Core script . . . . .	13
4.2	Interface . . . . .	14
4.2.1	Overall user interface . . . . .	14
4.2.2	Toolbox . . . . .	15
4.2.3	Workspace . . . . .	16
4.3	Algorithms . . . . .	16
4.3.1	Type checking . . . . .	17
4.3.2	Normalisation . . . . .	19
4.3.3	Modulus of continuity . . . . .	20
4.3.4	Examples . . . . .	21
<b>5</b>	<b>Implementation</b>	<b>24</b>
5.1	Setup . . . . .	24
5.1.1	Installations . . . . .	25
5.1.2	Execution . . . . .	25
5.2	Core backend script . . . . .	25
5.2.1	Type checking . . . . .	28
5.2.2	Normalisation . . . . .	28
5.2.3	Modulus of continuity . . . . .	29
5.3	Flow of program . . . . .	31

5.3.1	Defining Blockly blocks . . . . .	33
5.3.2	Defining a workspace (using the Blockly Developer Tools) . . . . .	35
5.3.3	Defining generators for block . . . . .	36
5.3.4	Headers for request . . . . .	37
5.3.5	Parsers supplied to Aeson . . . . .	38
5.3.6	Encoders supplied to Aeson . . . . .	39
5.3.7	Headers for response . . . . .	42
5.3.8	Displaying using MathML . . . . .	42
5.4	Examples . . . . .	45
<b>6</b>	<b>Testing</b>	<b>49</b>
<b>7</b>	<b>Conclusions</b>	<b>55</b>
7.1	Successes . . . . .	55
7.2	Issues . . . . .	55
7.2.1	Problems with frontend-backend interaction . . . . .	55
7.2.2	Complication in the code for the b-translation function . . . . .	56
7.2.3	Problems with Typesetting . . . . .	56
7.2.4	Problems with Blockly . . . . .	56
7.2.5	Suboptimal results . . . . .	56
7.3	Extensions and Improvements . . . . .	57
7.4	Overall conclusions . . . . .	57
	<b>Bibliography</b>	<b>58</b>
<b>A</b>	<b>Code</b>	<b>61</b>
A.1	File: core.hs . . . . .	62
A.2	File: server.js . . . . .	74
A.3	File: blocks.js . . . . .	75
A.4	File: index.html . . . . .	78
<b>B</b>	<b>Evidence of tests</b>	<b>83</b>
<b>C</b>	<b>Impact of COVID-19</b>	<b>103</b>

# List of Figures

2.1	Mikrokosmos . . . . .	9
4.1	Overall user interface . . . . .	14
4.2	Toolbox . . . . .	15
4.3	Workspace . . . . .	16
5.1	Flow of data . . . . .	31
5.2	Two components of the Blockly Developer Tools . . . . .	33
5.3	The three groups of blocks which were constructed using the Blockly Developer Tools . . . . .	33
5.4	Behaviour of the mutator of the context block . . . . .	34
5.5	Workspace factory . . . . .	35
5.6	Inputting the addition example into the tool . . . . .	45
5.7	Outputting the result of the addition example from the tool . . . . .	45
5.8	Inputting the addition example into the tool . . . . .	46
5.9	Outputting the result of the addition example from the tool . . . . .	46
5.10	Inputting the exponentiation example into the tool . . . . .	47
5.11	Outputting the result of the exponentiation example from the tool . . . . .	47
5.12	Inputting the modulus of continuity example into the tool . . . . .	48
5.13	Outputting the result of the addition example from the tool . . . . .	48
7.1	Typesetting comparison between MathML and MathJax . . . . .	56

# List of Tables

2.1	Comparison between the input language and the output language . . . . .	3
3.1	Functional requirements . . . . .	11
3.2	Non-functional requirements . . . . .	12
5.1	Versions of software . . . . .	24
5.2	Explanation of how types and the corresponding ADT relate . . . . .	25
5.3	Explanation of how terms and the corresponding ADT relate . . . . .	26
5.4	Explanation of how type errors and the corresponding ADT relate . . . . .	27
5.5	Explanation of how contexts and the corresponding ADT relate . . . . .	27
5.6	Explanation of how JSON strings are generated for terms . . . . .	36
5.7	Explanation of how JSON strings are generated for types . . . . .	36
5.8	Explanation of how terms are parsed from JSON strings . . . . .	38
5.9	Explanation of how types are parsed from JSON strings . . . . .	38
5.10	Explanation of JSON strings are encoded for types . . . . .	39
5.11	Explanation of how JSON strings are encoded for terms . . . . .	40
5.12	Explanation of how JSON strings are encoded for type errors . . . . .	41
5.13	Headers in response . . . . .	42
5.14	Explanation of how JSON is converted to MathML for types . . . . .	42
5.15	Explanation of how JSON is converted to MathML for terms . . . . .	43
5.16	Explanation of how JSON is converted to HTML for type errors . . . . .	44
6.1	Test plan . . . . .	52
B.1	Evidence of testing . . . . .	83



# List of Code Listings

4.1	Pseudocode for the definition of the output language . . . . .	16
4.2	Pseudocode for the definition of the input language . . . . .	16
4.3	Pseudocode for type checking . . . . .	17
4.4	Pseudocode for reductions . . . . .	19
4.5	Pseudocode for modulus of continuity . . . . .	20
4.6	Tracing pseudocode for type checking . . . . .	21
4.7	Tracing pseudocode for normalisation (addition) . . . . .	22
4.8	Tracing pseudocode for normalisation (exponentiation) . . . . .	23
4.9	Tracing pseudocode for b-translation . . . . .	23
5.1	Definition of the type ADT in the core script . . . . .	25
5.2	Definition of the term ADT in the core script . . . . .	26
5.3	Definition of the type error ADT in the core script . . . . .	27
5.4	Definition of the context type in the core script . . . . .	27

# Acknowledgements

I would like to thank my supervisor, Thomas Powell, his consistently constructive feedback and helpful criticism. I would also like to thank my parents for understanding my late nights, and my friends (specifically Awen Rhys, and Andrew Morton) for always listening to me complaining about  $\text{\LaTeX}$  and JavaScript, and Bence Babrián for redirecting me when I was very stuck.

# Chapter 1

## Introduction

The aim of this project is to develop a tool that gives the user a graphical user interface, that will compute inductive translations on terms in a  $\lambda$ -calculus. The specific  $\lambda$ -calculus that is being looked at is the typed  $\lambda$ -calculus, System  $T$  and the specific type of user interface being presented is a block-based visual programming language user interface.

System  $T$  is “an extension of the simply typed  $\lambda$ -calculus with numbers and a recursion operator” (Alves et al., 2010). Numbers are constructed by applying the successor constant  $\text{succ}$   $n$  times to the zero constant to represent  $n$ , and recursion occurs using the recursor constant  $\text{rec}$ . “It is a very simple system, yet has an enormous expressive power.” (Alves et al., 2010)

I could not find a  $\lambda$ -calculus tool built to either interpret terms in System  $T$ , or to use a block-based visual programming language interface: in either respect, my project is somewhat original. There are however existing text-based tools to interpret terms in the untyped  $\lambda$ -calculus. The topic of  $\lambda$ -calculus interpreter it is not new, but the approach to the interface is somewhat original. The topic of moduli of continuity in System  $T$  is also not new, but the method of computation (using b-translation) is taken from a relatively recent paper.

The tool will use the Blockly interface (written in HTML, CSS, JavaScript) to encode and display terms (and contexts) and it will use the b-translation defined by (Xu, 2020) to compute moduli of continuity.

This dissertation is organised into a literature and technology survey (where the existing literature, and similar existing tools are considered), requirements (where the software that has been developed is compared to the original thinking I had at the proposal stage), design (where algorithmic and interface designs are discussed), implementation (where the actual code and interface are discussed) and finally conclusions (where what was successful and what was less so is discussed).

## Chapter 2

# Literature and Technology Survey

In this chapter, the theory of the specific  $\lambda$ -calculus in question will be explained and other similar tools will be examined. Each of these will be discussed in turn.

### 2.1 Theoretical underpinning

The  $\lambda$ -calculus being used will be defined, System  $T$ , which is constrained to give the input language, and then is also extended to give the output language and then inductive translations on this  $\lambda$ -calculus are defined.

#### 2.1.1 The Ordinary System $T$

The ordinary System  $T$  is a typed  $\lambda$ -calculus first defined by (Gödel, 1958). The syntax of types in System  $T$  is given by

$$\tau ::= \mathbb{N} \mid \tau \rightarrow \tau \mid \tau \times \tau.$$

For reference, there is an explanation of how each type construction will be referred to. Every term  $m$  in System  $T$  is associated with a type  $\rho$  - this is denoted as  $m : \rho$ . Given types  $\sigma, \tau$ .

- $\mathbb{N}$  is a type 'naturals',
- $\sigma \rightarrow \tau$  is a type 'arrow',
- $\sigma \times \tau$  is a type 'product'.

Products are not available in the input language so they are only present in the output language. The syntax of terms in System  $T$  are given by

$$m ::= x \mid \lambda(x : \rho) . m \mid m m \mid \langle m; m \rangle \mid 0 \mid \text{succ} \mid \text{rec}_\rho.$$

For reference, there is an explanation of how each term construction will be referred to. Given types  $\tau, \sigma$  where  $m : \tau, n : \sigma, m' : \tau \rightarrow \sigma$ :

- $x : \tau$  is a term, 'variable'.
- $\lambda(x : \tau) . n : \tau \rightarrow \sigma$  is a term, 'abstraction' where  $x$  is a variable bound by this abstraction and  $m$  is the body of this abstraction.
- $m' n : \sigma$  is a term 'application' where  $m'$  is the 'function' and  $n$  the 'argument',
- $\langle m; n \rangle : \tau \times \sigma$  is a term 'pair' where  $m$  and  $n$  are its left and right components.

Pairs are not available in the input language so they are only present in the output language. Given any pair of the following:

- $n : \tau$ ,
- $m : \tau \rightarrow \sigma$ ,
- $m n : \sigma$ ,

the third can be deduced.

In addition to the terms above (variables, abstractions, applications, pairs) which are known as function types in (Xu, 2020), there are five constant terms defined as

- $0 : \mathbb{N}$ , “zero”;
- $\text{succ} : \mathbb{N} \rightarrow \mathbb{N}$ , “successor”;
- $\text{rec}_\rho : \rho \rightarrow (\mathbb{N} \rightarrow \rho \rightarrow \rho) \rightarrow \mathbb{N} \rightarrow \rho$  for each type  $\rho$  “recursor”;
- $\pi_1 : \sigma \times \tau \rightarrow \sigma$ ,  $\pi_2 : \sigma \times \tau \rightarrow \tau$ , for each type  $\sigma, \tau$ , “projections”.

### 2.1.2 Comparison between the input language and the output language

There are several constructions that are specific to the output language. The output language extends the input language in some important ways: the input language lacks projections, pairs, products.

For  $t : ((\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}) \times ((\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N})$ , two special projections (which inherit the properties of general projections) are defined by:

$$\begin{aligned} V_t &: (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}, \\ M_t &: (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}. \end{aligned}$$

where  $V_t = \pi_1(t)$ , known as the ‘value’ of  $t$ , and  $M_t = \pi_2(t)$ , known as the ‘modulus’ of  $t$ , giving the identity  $t = \langle V_t, M_t \rangle$  (Xu, 2020). The maximum denoted as  $\max : (\mathbb{N} \times \mathbb{N}) \rightarrow \mathbb{N}$  gives the greatest of two natural numbers.

Pairs, values, moduli and maximums are not available in the input language and so they are only present in the output language. For the sake of clarity, the difference between the input language and the output language should be made clear.

Table 2.1: Comparison between the input language and the output language

	Both input and output languages	Only output language
<i>Types</i>	<ul style="list-style-type: none"> <li>• Naturals</li> <li>• Arrow</li> </ul>	<ul style="list-style-type: none"> <li>• Product</li> </ul>
<i>Terms</i>	<ul style="list-style-type: none"> <li>• Zero,</li> <li>• Successor,</li> <li>• Recursor,</li> <li>• Variable,</li> <li>• Application,</li> <li>• Abstraction</li> </ul>	<ul style="list-style-type: none"> <li>• Pair,</li> <li>• Projection,</li> <li>• Value,</li> <li>• Modulus,</li> <li>• Maximum</li> </ul>

### 2.1.3 Inductive functions in System $T$

Before delving into more complicated inductive functions, there first needs to be an explanation as to what the basic idea is and give an example.

An inductive translation can be defined inductively on variables, applications and abstractions, zero, the successor, and the recursor and therefore be defined on all terms in the input language of System  $T$  by induction. A simple inductive function is the set of all free variables of a term  $t \mapsto \text{FV}(t) := \{x_1, \dots, x_n\}$ .

In System  $\mathcal{T}$ , FV is given by as

$$\begin{aligned} \text{FV}(x) &:= \{x\}, \\ \text{FV}(\lambda(x : \rho). m) &:= \text{FV}(m) \setminus \{x\}, \quad \forall \rho, \\ \text{FV}(m n) &:= \text{FV}(m) \cup \text{FV}(n), \\ \text{FV}(0) &:= \emptyset, \\ \text{FV}(\text{succ}) &:= \emptyset, \\ \text{FV}(\text{rec}_\rho) &:= \emptyset \quad \forall \rho \end{aligned}$$

in the input language.

### Formal typing rules

The *valid type judgements* are generated by the following rules.

$$\begin{aligned} &\frac{\Gamma \vdash m : \tau \rightarrow \sigma \quad \Gamma \vdash n : \tau}{\Gamma \vdash m n : \sigma} (app) \\ &\frac{\Gamma, x : \rho \vdash m : \sigma}{\Gamma \vdash \lambda(x : \rho). m : \rho \rightarrow \sigma} (abs) \\ &\frac{x_1 : \tau_1, \dots, x_n : \tau_n \vdash x_i : \tau_i}{\vdash \text{rec}_\rho : \rho \rightarrow (\mathbb{N} \rightarrow \rho \rightarrow \rho) \rightarrow \mathbb{N} \rightarrow \rho} \\ &\vdash 0 : \mathbb{N} \\ &\vdash \text{succ} : \mathbb{N} \rightarrow \mathbb{N} \end{aligned}$$

The following rules are only in the output language.

$$\begin{aligned} &\frac{\Gamma \vdash t : ((\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}) \times ((\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N})}{\Gamma \vdash V_t : (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}} (val) \\ &\frac{\Gamma \vdash t : ((\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}) \times ((\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N})}{\Gamma \vdash V_t : (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}} (mod) \\ &\frac{\Gamma \vdash m : \sigma \times \tau}{\Gamma \vdash \pi_1 m : \sigma} (proj-1) \\ &\frac{\Gamma \vdash m : \sigma \times \tau}{\Gamma \vdash \pi_2 m : \tau} (proj-2) \\ &\frac{\Gamma \vdash m : \mathbb{N} \quad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \max(m, n) : \mathbb{N}} (max) \\ &\frac{\Gamma \vdash m : \tau \quad \Gamma \vdash n : \sigma}{\Gamma \vdash \langle m, n \rangle : \tau \times \sigma} (pair) \end{aligned}$$

### $\alpha$ -equivalence in System $\mathcal{T}$

Two  $\lambda$ -terms are considered to be equivalent if they only differ by the names of their bound variables, but the types of the variables are the same.

$\alpha$ -conversion is the process of renaming bound variables in terms. For any term  $m$  and variables  $x, y$ , ' $m$  with every occurrence of  $x$  replaced with  $y$ ' is denoted as  $m[y/x]$ . A requirement is that  $x$  and  $y$  are the same type. If  $m$  and  $n$  are  $\alpha$ -equivalent then  $m$  and  $n$  are identical up to the choice of names for bound variables, and is written as  $m =_\alpha n$ .

### $\eta$ -reduction in System $T$

$\eta$ -reduction is a standard reduction and simply eliminates ‘redundant functions’.  $\eta$ -redexes are of the form  $\lambda(x : \rho) . m x$  which correspond to redundant functions. The operation which is described as “removing redundant abstractions”:

$$\lambda(x : \rho) . m x \rightarrow_{\eta} m. \quad (2.1)$$

The notation  $m \rightarrow_{\eta}^* m'$  is used to mean that a term  $m$   $\eta$ -reduces to another term  $m'$  in a number of steps (this would be zero steps if the term were already fully  $\eta$ -reduced). A term is said to have been fully  $\eta$ -reduced when there are no  $\eta$ -reducible expressions left. Two terms  $m$  and  $n$  are considered  $\eta$ -equivalent if  $m^* =_{\alpha} n^*$  where  $m^*$  and  $n^*$  are the fully  $\eta$ -reduced forms of  $m, n$  respectively.

### $\beta$ -reduction in System $T$

The next most simple reduction is  $\beta$ -reduction and simply calls to a ‘function call’ in programming.  $\beta$ -redexes are of the form  $(\lambda x . m) n$  which correspond to applying a function to a given argument. This stops when there are no redexes left. The operation which is described by a “single step of computation” is denoted as

$$(\lambda(x : \rho) . m) n \rightarrow_{\beta} m[n/x], \quad (2.2)$$

where  $x$  is not free in  $m$ . Without this condition, this could result in capture of free variables.

In the above definition  $m[n/x]$  means “ $m$  with  $n$  substituted for every free occurrence of  $x$ ”. As well as this,  $x$  and  $n$  must be of the same type, for this to be valid. Formally this is defined as:

$$\begin{aligned} y[n/x] &:= \begin{cases} n & y = x \\ y & y \neq x \end{cases} \\ (\lambda(x : \rho) . m)[n/x] &:= \begin{cases} \lambda(x : \rho) . m & x \notin \text{FV}(m), \\ \lambda(x : \rho) . m[z/x][n/y] & x \in \text{FV}(m), \end{cases} \quad \forall \rho; \\ (m n)[n/x] &:= (m[n/x]) (n[n/x]); \\ \text{succ}[n/x] &:= \text{succ}; \\ \text{rec}_{\rho}[n/x] &:= \text{rec}_{\rho}, \quad \forall \rho; \\ 0[n/x] &:= 0. \end{aligned}$$

The notation  $m \rightarrow_{\beta}^* m'$  to mean that a term  $m$   $\beta$ -reduces to another term  $m'$  in a number of steps (possibly zero if it was already fully  $\beta$ -reduced). A term is said to have been fully  $\beta$ -reduced when there are no  $\beta$ -reducible expressions left. Two terms  $m$  and  $n$  are considered  $\beta$ -equivalent if  $m^* =_{\alpha} n^*$  where  $m^*$  and  $n^*$  are the fully  $\beta$ -reduced forms of  $m, n$  respectively.

### Defining recursion using reductions

System  $T$  allows for recursion, using  $\text{rec}$ . Recursion in System  $T$  is defined using two rules.

$$\text{rec}_{\rho}(a)(f)(0) \rightarrow a, \quad (2.3)$$

$$\text{rec}_{\rho}(a)(f)(\text{succ } n) \rightarrow f(n)(\text{rec}_{\rho}(a)(f)(n)). \quad (2.4)$$

The first rule is a ‘base case’ and the second rule is a ‘recursive case’. This definition of recursion is equivalent to non-tail recursion (Muchnick, 1998).

### Defining projections using reductions

Projections allow for the extraction of a component from a pair. The equation  $t := \langle \pi_1 t, \pi_2 t \rangle$  gives the reduction rule

$$\langle \pi_1 t, \pi_2 t \rangle \rightarrow t$$

Let  $t$  be defined as  $\langle p; q \rangle$ . In line with the definitions of  $\pi_1$  and  $\pi_2$ ,

$$\pi_1 t \rightarrow p,$$

$$\pi_2 t \rightarrow q.$$

Projections are not available in the input language so they are only present in the output language.

### Additional reductions

There are six further reduction rules only defined in the output language. The equation  $t := \langle V_t; M_t \rangle$  gives the reduction rule

$$\langle V_t; M_t \rangle \rightarrow t. \quad (2.5)$$

Let  $t$  be defined as  $\langle p; q \rangle$ . In line with the definitions of  $V_t$  and  $M_t$ ,

$$V_t \rightarrow p, \quad (2.6)$$

$$M_t \rightarrow q. \quad (2.7)$$

The maximum function could be defined using  $\text{rec}$  (Xu, 2020) but it will instead be governed by the following four reduction rules:

$$\max(\text{succ } i, \text{succ } j) \rightarrow \text{succ } \max(i, j), \quad (2.8)$$

$$\max(i, 0) \rightarrow i, \quad (2.9)$$

$$\max(0, i) \rightarrow i, \quad (2.10)$$

$$\max(i, i) \rightarrow i \quad (2.11)$$

where  $i : \mathbb{N}, j : \mathbb{N}$ . It should be noted that (2.11) is important for cases involving free variables: if there were no free variables, it could be derived from the other rules.

### Defining normalisation in System $T$

'Reducible' expressions ("redexes") are defined to include those that could be  $\beta$ -reduced,  $\eta$ -reduced or reduced by any other reduction rule from any of the numbered rules that have been introduced (the ten numbered rules constitute the list of reduction rules). From that, the definition of a normal form changes to mean that there are no redexes, except by  $\alpha$ -redexes, left. Two terms  $m$  and  $n$  are considered equivalent if  $m^* =_\alpha n^*$  where  $m^*$  and  $n^*$  are the normal forms of  $m, n$  respectively. Uniqueness of the normal form of a term in System  $T$  is ensured by the Church-Rosser Theorem. (2.2.28 and 2.2.33 in (Troelstra, 1973))

#### 2.1.4 Continuity in System $T$

A function  $f : (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$  is *continuous* if for any sequence  $\alpha : \mathbb{N} \rightarrow \mathbb{N}$ , there exists  $m : \mathbb{N}$  called the *point of continuity* at the point  $\alpha$ , such that any sequence  $\beta : \mathbb{N} \rightarrow \mathbb{N}$  that is equal to  $\alpha$  up to the first  $m$  positions gives the same result.

Every term in System  $T$  is continuous, a proof of this is given by (Xu, 2020), but this is a well-known property.

In an equational calculus, the modulus of continuity  $M : (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$  of  $f$  is defined such that:

$$\alpha =_M(\alpha) \beta \implies f(\alpha) = f(\beta)$$

for all  $\alpha : \mathbb{N} \rightarrow \mathbb{N}, \beta : \mathbb{N} \rightarrow \mathbb{N}$ , where  $\alpha =_n \beta$  means that  $\alpha i = \beta i$  for all  $i < n$ .

The modulus of continuity of a term is the main computation and focus of this project. This necessitates a translation known as the "b-translation" of a term and defined in (Xu, 2020), although other methods of computing the modulus of continuity also exist. The rest of this project will be developed in an equivalent reductional  $\lambda$ -calculus.

#### 2.1.5 b-translation

The b-translation is a translation that will be used to find the modulus of continuity. Only the definition of b-translation will be presented - full details can be found in (Xu, 2020). For each type  $\rho$  in the input language, a type  $\rho^b$  is associated inductively as follows:

$$\begin{aligned} \mathbb{N}^b &:= ((\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}) \times ((\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}), \\ (\sigma \rightarrow \tau)^b &:= \sigma^b \rightarrow \tau^b, \end{aligned}$$

in the output language. The type,  $((\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}) \times ((\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N})$  is denoted as  $\mathbb{N}^b$  for the sake of simplicity.



An inductive function, “b-translation”  $t \mapsto t^b$  can be defined below.

A mapping of variables  $x : \rho$  to variables  $x^b : \rho^b$  can be defined, where  $\rho^b$  is defined inductively above ( $\star$ ).

For any term  $t : \rho$  in the input language of System  $T$ ,  $t^b : \rho^b$  can be defined in the output language of System  $T$  as follows:

$$\begin{aligned} (x)^b &:= x^b, \\ (\lambda(x : \rho) . n)^b &:= \lambda(x^b : \rho^b) . n^b, \\ (m n)^b &:= m^b n^b, \\ 0^b &:= \langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . 0; \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . 0 \rangle, \\ \text{succ}^b &:= \lambda(x : \mathbb{N}^b) . \langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ}(\mathbf{V}_x(\alpha)); \mathbf{M}_x \rangle, \\ \text{rec}_\rho^b &:= \lambda(a : \rho^b) . \lambda(f : \mathbb{N}^b \rightarrow (\rho^b \rightarrow \rho^b)) . \lambda(h : \mathbb{N}^b) . \\ &\quad \mathbf{ke}_\rho(\text{rec}_{\rho^b}(a)(\lambda(k : \mathbb{N}) . f \langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . k; \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . 0 \rangle))(h), \end{aligned}$$

where the Kleisli extension  $\mathbf{ke}_\rho : (\mathbb{N} \rightarrow \rho^b) \rightarrow \mathbb{N}^b \rightarrow \rho^b$  is defined by

$$\begin{aligned} \mathbf{ke}_\mathbb{N}(g)(f) &= \langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \mathbf{V}_g(\mathbf{V}_f(\alpha))(\alpha); \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \max(\mathbf{M}_{g(\mathbf{V}_f(\alpha))}(\alpha), \mathbf{M}_f(\alpha)) \rangle, \\ \mathbf{ke}_{\sigma \rightarrow \tau}(g)(f) &= \lambda(x : \sigma^b) . \mathbf{ke}_\tau(\lambda(k : \mathbb{N}) . g(k)(x))(f). \end{aligned}$$

The types have all been made explicit, which is not the case in (Xu, 2020). It should be noted that the recursion reduction rules apply in a similar way to  $\text{rec}^b$ , where these rules are equivalent to using combinations of other rules:

$$\begin{aligned} \text{rec}_\rho^b(a)(f)(0^b) &\rightarrow a, \\ \text{rec}_\rho^b(a)(f)((\text{succ } n)^b) &\rightarrow f(n^b)(\text{rec}_\rho^b(a)(f)(n^b)). \end{aligned}$$

These derived rules appear in (Xu, 2020). Let  $\Omega$  be defined as

$$\lambda(f : \mathbb{N}^b) . \langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \alpha(\mathbf{V}_f(\alpha)); \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \max(\mathbf{M}_f(\alpha), \text{succ } \mathbf{V}_f(\alpha)) \rangle.$$

The modulus of continuity of  $f : (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$  is equal to  $\mathbf{M}_{t^b \Omega}$ . This is proved in Section 4.1 of (Xu, 2020) but the details will not be discussed. The only thing that matters is that the b-translation will be implemented so that  $\mathbf{M}_{t^b \Omega}$  can be computed from an input term.

### 2.1.6 Examples

These four examples use the three main computations that the user can use. They will be examined further to show how the designed algorithms would work and how the tool actually works.

#### Type checking

The type checking example Given a term  $((\text{rec}_\mathbb{N} x)(\lambda(u : \mathbb{N}) . \text{succ})) 0$  (with context  $\Gamma$  given as  $x : \mathbb{N}$ ), its type can be deduced.

$$\frac{\frac{\Gamma \vdash \text{rec}_\mathbb{N} : \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N} \rightarrow \mathbb{N} \quad \Gamma \vdash x : \mathbb{N}}{\Gamma \vdash \text{rec}_\mathbb{N} x : (\mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N} \rightarrow \mathbb{N}}(app) \quad \frac{\Gamma, u : \mathbb{N} \vdash \text{succ} : \mathbb{N} \rightarrow \mathbb{N}}{\Gamma \vdash \lambda(u : \mathbb{N}) . \text{succ} : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}}(abs)}{\Gamma \vdash ((\text{rec}_\mathbb{N} x)(\lambda(u : \mathbb{N}) . \text{succ})) : \mathbb{N} \rightarrow \mathbb{N}}(app) \quad \frac{\Gamma \vdash 0 : \mathbb{N}}{\Gamma \vdash ((\text{rec}_\mathbb{N} x)(\lambda(u : \mathbb{N}) . \text{succ})) 0 : \mathbb{N}}(app)$$

This term was chosen because it uses all six constructions of terms that are in the input language.

#### Normalisation: Addition

The term given by

$$\lambda(x : \mathbb{N}) . \lambda(y : \mathbb{N}) . \text{rec}_\mathbb{N}(x)(\lambda(u : \mathbb{N}) . \lambda(v : \mathbb{N}) . \text{succ}(v))(y)$$

is the addition operator. Adding two numerals using this term can be considered. Given the example term:

$$(\lambda(x : \mathbb{N}) . \lambda(y : \mathbb{N}) . \text{rec}_\mathbb{N}(x)(\lambda(u : \mathbb{N}) . \lambda(v : \mathbb{N}) . \text{succ}(v))(y))(\text{succ}(\text{succ}(\text{succ } 0)))(\text{succ}(\text{succ } 0)),$$

can be normalised as:

$$\begin{aligned}
& (\lambda(x : \mathbb{N}) . \lambda(y : \mathbb{N}) . \text{rec}_{\mathbb{N}}(x) (\lambda(u : \mathbb{N}) . \lambda(v : \mathbb{N}) . \text{succ}(v)) (y)) (\text{succ}(\text{succ}(\text{succ } 0))) (\text{succ}(\text{succ } 0)) \\
\rightarrow & (\lambda(x : \mathbb{N}) . \text{rec}_{\mathbb{N}}(x) (\lambda(u : \mathbb{N}) . \text{succ})) (\text{succ}(\text{succ}(\text{succ } 0))) (\text{succ}(\text{succ } 0)) \\
\rightarrow & (\text{rec}_{\mathbb{N}}(\text{succ}(\text{succ}(\text{succ } 0))) (\lambda(u : \mathbb{N}) . \text{succ})) (\text{succ}(\text{succ } 0)) \\
\rightarrow & \text{succ}(\text{rec}_{\mathbb{N}}(\text{succ}(\text{succ}(\text{succ } 0))) (\lambda(u : \mathbb{N}) . \text{succ})) (\text{succ } 0) \\
\rightarrow & \text{succ}(\text{succ}(\text{rec}_{\mathbb{N}}(\text{succ}(\text{succ}(\text{succ } 0))) (\lambda(u : \mathbb{N}) . \text{succ})) (0)) \\
\rightarrow & \text{succ}(\text{succ}(\text{succ}(\text{succ}(\text{succ } 0)))) .
\end{aligned}$$

### Normalisation: Exponentiation

The term given as

$$\lambda(x : \mathbb{N}) . (\text{rec}_{\mathbb{N}}(\text{succ } 0)) (\lambda(z : \mathbb{N}) . (\text{rec}_{\mathbb{N}} 0) (\lambda(y : \mathbb{N}) . (\text{rec}_{\mathbb{N}} x) (\lambda(u : \mathbb{N}) . \text{succ})))$$

is the exponentiation operator - one interesting question is to ask what happens when the term corresponding to  $0^0$ , which is invalid in conventional arithmetic, can be computed. Given the term:

$$(\lambda(x : \mathbb{N}) . (\text{rec}_{\mathbb{N}}(\text{succ } 0)) (\lambda(z : \mathbb{N}) . (\text{rec}_{\mathbb{N}} 0) (\lambda(y : \mathbb{N}) . (\text{rec}_{\mathbb{N}} x) (\lambda(u : \mathbb{N}) . \text{succ})))) (0) (0),$$

can normalised as the following:

$$\begin{aligned}
& (\lambda(x : \mathbb{N}) . (\text{rec}_{\mathbb{N}}(\text{succ } 0)) (\lambda(z : \mathbb{N}) . (\text{rec}_{\mathbb{N}} 0) (\lambda(y : \mathbb{N}) . (\text{rec}_{\mathbb{N}} x) (\lambda(u : \mathbb{N}) . \text{succ})))) (0) (0) \\
\rightarrow & (\text{rec}_{\mathbb{N}}(\text{succ } 0)) (\lambda(z : \mathbb{N}) . (\text{rec}_{\mathbb{N}} 0) (\lambda(y : \mathbb{N}) . (\text{rec}_{\mathbb{N}} 0) (\lambda(u : \mathbb{N}) . \text{succ}))) (0) \\
\rightarrow & \text{succ } 0.
\end{aligned}$$

### Modulus of continuity

Given the term:  $t := \lambda(a : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ}(a(a\ 0))$  (a simple example), its modulus of continuity is given by:  $M_{t^b} \Omega$  meaning the first task to complete, would be to compute the b-translation of  $t$ . The b-translation of  $t$  can be computed as

$$\begin{aligned}
t^b &= (\lambda(a : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ}(a(a\ 0)))^b \\
&= (\lambda(a^b : (\mathbb{N} \rightarrow \mathbb{N})^b) . (\text{succ}(a(a\ 0)))^b)^b \\
&= (\lambda(a^b : \mathbb{N}^b \rightarrow \mathbb{N}^b) . \text{succ}^b(a(a\ 0)))^b \\
&= \lambda(a^b : \mathbb{N}^b \rightarrow \mathbb{N}^b) . \text{succ}^b(a^b(a\ 0))^b \\
&= \lambda(a^b : \mathbb{N}^b \rightarrow \mathbb{N}^b) . \text{succ}^b(a^b(a\ 0)^b) \\
&= \lambda(a^b : \mathbb{N}^b \rightarrow \mathbb{N}^b) . \text{succ}^b(a^b(a^b\ 0^b)) \\
&= \lambda(a^b : \mathbb{N}^b \rightarrow \mathbb{N}^b) . (\lambda(x : \mathbb{N}^b) . \langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ}(V_x(\alpha)); M_x \rangle) (a^b(a^b(\langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . 0; \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . 0 \rangle))) \\
\rightarrow^* & \lambda(a^b : \mathbb{N}^b \rightarrow \mathbb{N}^b) . \langle \lambda(d : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ}(V_{a^b(a^b\ \langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . 0; \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . 0 \rangle}) d); M_{a^b(a^b\ \langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . 0; \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . 0 \rangle)} \rangle,
\end{aligned}$$

which gives us  $M_{t^b} \Omega$  as

$$\begin{aligned}
M_{t^b} \Omega &=^* M_{\langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ}(V_{\Omega(\Omega\ \langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . 0; \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . 0 \rangle}) \alpha \rangle; M_{\Omega(\Omega\ \langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . 0; \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . 0 \rangle)}} \\
&\rightarrow^* M_{\Omega(\Omega\ \langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . 0; \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . 0 \rangle)}.
\end{aligned}$$

Note that

$$\begin{aligned}
& \Omega \langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . 0; \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . 0 \rangle \\
&= \lambda(f : \mathbb{N}^b) . \langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \alpha(V_f(\alpha)); \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \max(M_f(\alpha), \text{succ } V_f(\alpha)) \rangle \langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . 0; \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . 0 \rangle \\
\rightarrow^* & \langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \alpha(0); \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \max(0, \text{succ } 0) \rangle \\
\rightarrow & \langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \alpha(0); \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ } 0 \rangle,
\end{aligned}$$

and therefore

$$\begin{aligned}
& \Omega (\Omega \langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . 0; \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . 0 \rangle) \\
&= \Omega \langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \alpha (0); \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ } 0 \rangle \\
&\rightarrow^* \langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \alpha (\alpha 0); \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \max(\text{succ } 0, \text{succ } \alpha (0)) \rangle \\
&\rightarrow \langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \alpha (\alpha 0); \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ } \max(0, \alpha (0)) \rangle \\
&\rightarrow \langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \alpha (\alpha 0); \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ } (\alpha (0)) \rangle.
\end{aligned}$$

When this is substituted to finally find the normal form of  $M_{t^b} \Omega$ , the result can be seen to be

$$\begin{aligned}
M_{t^b} \Omega &=^* M_{\langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \alpha (\alpha 0); \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ } (\alpha (0)) \rangle} \\
&\rightarrow \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ } (\alpha 0),
\end{aligned}$$

which is in normal form. The modulus of continuity of  $t$  is therefore  $\lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ } (\alpha 0)$ .

## 2.2 Existing tools

Before designing any tool, existing tools need to be reviewed. In this case, there are few projects that are similar enough to be considered. The Agda implementation of b-translation and an existing graphical  $\lambda$ -calculus tool for a typed  $\lambda$ -calculus will be examined, because they are similar enough, but in different respects to the tool that is being built here.

### 2.2.1 Agda implementation

The tool that is the most similar to the tool that has been developed here is the implementation of (Xu, 2020) in Agda is given by (Xu, 2019). The implementation currently lacks a graphical user interface (GUI). On top of that, Agda is difficult to use, particularly because of its heavy use of Unicode symbols that are beyond the scope of a typical keyboard (Wadler, Kokke and Siek, 2020).

### 2.2.2 Mikrokosmos

Mikrokosmos is a similar tool: it provides a graphical interface. Mikrokosmos is an educational  $\lambda$ -calculus tool that contains an interface for inputting typed terms in the untyped  $\lambda$ -calculus and the simply typed  $\lambda$ -calculus, but  $\lambda$ -terms are not input inductively: they are input by typing text. An interface that allows encoding terms in an inductive way would be better, because this is closer to how they are defined and thought of (see Section 1.2) (Román, 2019).

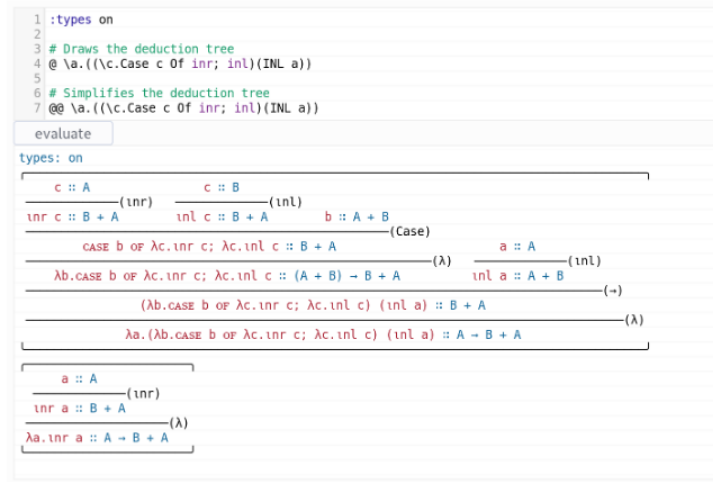


Figure 2.1: Mikrokosmos

# Chapter 3

## Requirements

In this chapter, there will be an explanation of how requirements have been met, not met, or discarded, and they will be fully listed.

### 3.1 Achieved requirements

In comparison to the requirements as described by the proposal, the vast majority of the requirements have been achieved.

Chapters 5 and 6 demonstrate how they were achieved and verify that this is the case.

### 3.2 Discarding of certain requirements

Certain requirements have been discarded because they were inappropriate, and are fully listed in Tables 3.1 and Table 3.2.

#### 3.2.1 Division between intermediate computations and final computation

The requirement to ensure a clear division between intermediate modulus of continuity computations and the final modulus of continuity computation was discarded, because there is only ever one modulus of continuity that is computed for a single term.

#### 3.2.2 Complexity

Requirements that relate to complexity as defined by (Danner, Paykin and Royer, 2013) have been discarded. This is because the complexity computation defined there would have been too time-consuming to implement on top of the modulus of continuity computation that has been implemented, because it would have made the project too large to complete in the time that was available.

#### 3.2.3 Hosting

The requirement that deals with making the project simple to use without running any additional software, was discarded, because that would have required hosting.

Hosting was deemed to be out of the scope of the project, because it was too time-consuming to complete, in the time that was available. It would be relatively simple to finish the implementation of the hosting, from what has already been developed here.

### 3.3 Lists of requirements

The full lists of requirements are given below.

Table 3.1: Functional requirements

Index	Description	Outcome	Detail
1	The system must provide an interface for input and output in terms of terms in System <i>T</i> .	Achieved	See below.
1.1	The system must provide an interface for accepting input in terms of $\lambda$ -terms from the user.	Achieved	See below.
1.1.1	The system must allow the encoding of $\lambda$ -terms as input.	Achieved	The Blockly interface handles this.
1.1.2	The system could provide an interface for displaying input $\lambda$ -terms as output to the user for verification purposes.	Achieved	This is achieved using the echo command.
1.2	There must be an interface for producing the translation output in terms of $\lambda$ -terms.	Achieved	See below.
1.2.1	The system should provide an interface for displaying $\lambda$ -terms as output.	Achieved	This is achieved using the toHTML function.
1.2.2	The system should provide a method of outputting the result of every computation that the system computes.	Discarded	See below
1.2.2.1	The system should provide a method of outputting the result of every modulus of continuity that the system computes.	Discarded (mis-understanding)	Only one modulus of continuity is ever computed.
1.2.2.1	The system could provide a method of outputting the result of every complexity analysis that the system computes.	Discarded	See 2.2
1.2.3	The system should provide an interface for displaying input $\lambda$ -terms to allow the user to verify that they input the term correctly.	Achieved	This has been achieved using the echo command.
2	The system must provide a Haskell subsystem to implement the translation subsystem.	Achieved	See below.
2.1	The system must provide an implementation of modulus of continuity (and therefore normalisation and type checking) (Xu, 2020).	Achieved	This has been achieved using the Haskell subsystem.
2.2	The system could provide an implementation of the complexity analysis (Danner, Paykin and Royer, 2013).	Discarded	Complexity features were discarded because it would have made the project too large.
3	The system must implement an API for the Haskell subsystem to interface with the JavaScript GUI subsystem.	Achieved	This is achieved by the Node.js server.
4	The system could display a friendly error message when the system detects an error.	Achieved	Errors are displayed as nested unordered lists using the TypeError ADT in the core file and the toHTML function.
5	The system could be colour-coded to make it easier to use.	Achieved	This has been achieved using the Blockly interface.

Table 3.2: Non-functional requirements

Index	Description	Outcome	Detail
1	The system should have a clear separation between any intermediate steps being output and the final computation being output.	Discarded (mis-understanding)	There are no intermediate steps (see 1.2.2.1).
2	The system must present all output by the system in a manner that can be easily under-stood by a user with some knowledge of the relevant theory.	Achieved	This is handled by the toHTML function and MathML to produce a typeset output.
3	The system should be portable and simple to set up and not require installing additional software to compile or interpret code.	Discarded	This would require hosting. There are technical issues with this (see Section 7.2.1). This was deemed beyond the scope of the project.
4	The system should be efficient and responsive with respect to user interaction with the interface.	Achieved	When it is run locally, it executes quickly.

# Chapter 4

## Design

In this chapter, there will be a discussion of my main planning decisions behind the implementation. There were two main problems to be solved: the design of the interface and how to design the algorithms for the backend. Each of these will be discussed in turn.

### 4.1 Initial thoughts

#### 4.1.1 Interface

Blockly blocks can be nested inside each other in an inductive way. Blockly blocks can assert an 'output type' to ensure that blocks are syntactically correct.

The system should be able to preemptively block a user placing a term block into a type gap and vice versa: deeper type checking will be handled by the core script. There need to be user-facing errors in case a modulus of gaps or empty variable names in a term, type, context. These are referred to as 'syntax errors', which occur in cases where a term is grammatically incorrect.

Terms that are only available in the output language do not need to be associated with any block because they cannot be input. By using Blockly, colour-coding is extremely simple.

To display terms and types, typical  $\lambda$ -calculus notation will be used so that the output will be understood by the user.

#### 4.1.2 Interaction

The core script and the interface need to communicate and JSON is suitable for this because it can be used to nest objects inside of other objects (JSON Schema, 2020).

#### 4.1.3 Core script

Unlike the untyped  $\lambda$ -calculus, terms need to have types. This project will not be using type inference, so the best way to achieve this would be to force the user to state the types of bound variables in the abstraction block, and to state the types of free variables in a context block. This means that there has to be an implementation of type blocks, and some way for the interface to tell the difference between term blocks and type blocks.

Terms and types must be encoded as algebraic data type, allowing for recursion, and would be relatively simple. Contexts are best encoded using strict finite maps, ensuring a quick retrieval and insertion. This is part of the standard library, and so it can be assumed to work as expected.

There needs to be some notion of 'encodability' to distinguish the input and output languages. While projections are technically speaking available in the output language, there is no need to use them beyond their use in modulus and value. They will not be implemented in the general case. This means that there are effectively only three constants available.

There need to be user-facing errors in case a modulus of continuity is attempted with a term not of type  $(\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$ , a free variable is not found in the context and finally the case of an error. These are referred

to as ‘runtime errors’ which occur in cases where the term is grammatically correct, but impossible to interpret. Other errors that are possible to cause from within the script, such as attempting to find a b-translation of term in the output language, such as a pair, can be entirely prevented from the interface side.

## 4.2 Interface

In this section, there will be a discussion of the various components of the interface: the workspace, the toolbox, the display area, and how they connect.

### 4.2.1 Overall user interface

Before, discussing any of the individual components, a top-down view of the overall interface needs to be described.

The main user interface consists of a workspace, a toolbox and a display area, with some buttons. The user would select blocks by clicking or selecting them from the toolbox, and move them to the workspace (by dragging) and then place them in the workspace by unclicking or dropping them. This mockup describes where the individual components would be placed. Before each component can be discussed in detail, how they connect needs to be shown.

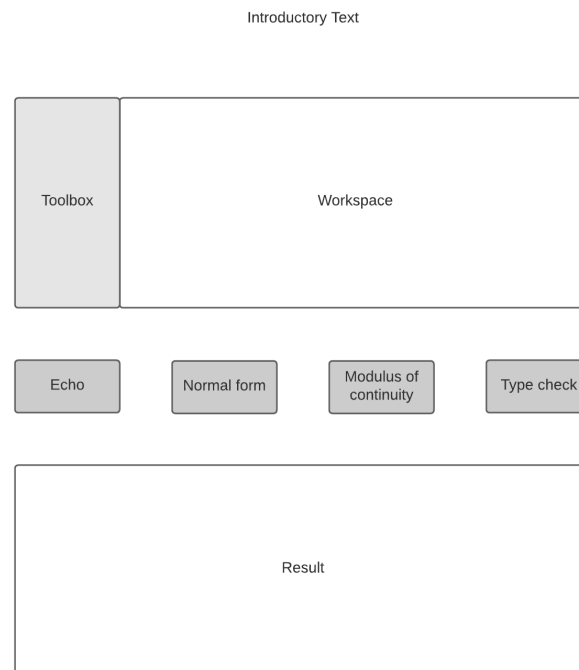


Figure 4.1: Overall user interface



### 4.2.2 Toolbox

The toolbox contains blocks that the user would select to place in the workspace.

The toolbox would contain two sections, for term blocks and type blocks. This is important, the group that each one belongs to indicates the output type of the block (Type or Term) and the gaps in the block each check that the block being inserted into a gap has an appropriate output type. Each gap in each block, has an indication of what blocks it would accept. This will mean that the process of parsing to Haskell will be easier. Wherever a type is expected, there will be a type, and wherever a term is expected there will be a term. Of course, 'Text' indicates that the user would type text into the text field - this is done to name variables, and occurs either in the variable block or in the abstraction block.

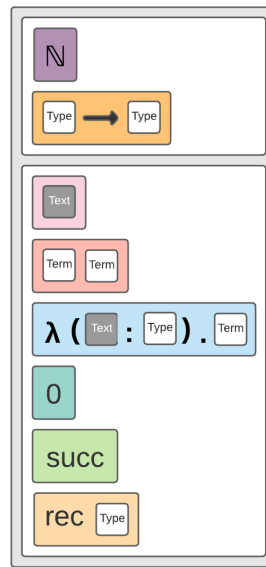


Figure 4.2: Toolbox

### 4.2.3 Workspace

The workspace is where the blocks are manipulated.

The workspace would initially contain the term parent block, in order to select which term is to be used for a computation, and the context block, in order to set the types of free variables. They are both necessary because of the following reasons.

- If there multiple orphan term blocks on the workspace, there needs to be a clear and obvious way to indicate which one is the intended to be used for a computation, and the best way to do this is to create a term parent block, and 'disable' orphan blocks. Orphan blocks are ignored by generators.
- Free variables' types need to be stated - this is done through the context block.

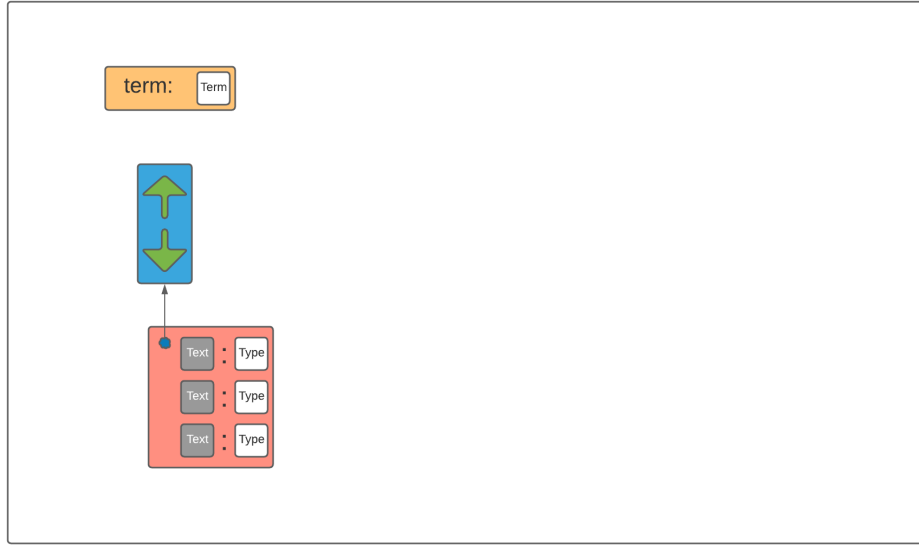


Figure 4.3: Workspace

## 4.3 Algorithms

Before the algorithms are defined, the syntax of the two languages needs to be explained.

The output language has the following definition.

```
Type := [ $\tau ::= \mathbb{N} \mid \tau \rightarrow \tau \mid \tau \times \tau$ ]
Context := [ $\Gamma ::= [x : \tau, \dots, x : \tau]$ ]
Term := [ $m ::= x \mid \lambda(x : \rho) . m \mid m \, m \mid 0 \mid \text{succ} \mid \text{rec}_\rho \mid \langle m; m \rangle \mid V_m \mid M_m \mid \max(m, m)$ ]
```

Code Listing 4.1: Pseudocode for the definition of the output language

The input language is a restriction of the output language.

```
(Type : Input) := [ $\tau ::= \mathbb{N} \mid (\tau : \text{Input}) \rightarrow (\tau : \text{Input})$ ]
(Context : Input) := [ $\Gamma ::= [x : (\tau : \text{Input}), \dots, x : (\tau : \text{Input})]$ ]
(Term : Input) := [ $m ::= x \mid \lambda(x : (\rho : \text{Input})) . (m : \text{Input}) \mid (m : \text{Input}) (m : \text{Input}) \mid 0 \mid \text{succ} \mid \text{rec}_{\rho : \text{Input}}$ ]
```

Code Listing 4.2: Pseudocode for the definition of the input language

### 4.3.1 Type checking

Type checking is necessary to ensure a term is not malformed, otherwise other more complicated translations cannot be computed. This is really important, because a term may be grammatically correct, i.e. there are no gaps, variable names are non-empty, and gaps that correspond to terms and types are filled by terms and types respectively, ensured by the interface, but the interface cannot make type deductions. This necessitates a specific type checking function to handle this work.

The type checking function can be represented using the following pseudocode.

```

TC :- Context → Term → Type
TC (Γ = [x1 : τ1, ..., xn : τn]) x =
  -- (var)
  | x ∈ Γ := Γ[x] -- look up x (key-value pairs, in this case: variable name-type pairs)
  | otherwise := Err
TC Γ (λ(x : ρ) . m) = ρ → TC ((x, ρ) : Γ) m
-- inserting (x : ρ) into Γ
TC Γ 0 := ℕ
TC Γ succ := ℕ → ℕ
TC Γ recρ := ρ → (ℕ → ρ → ρ) → ℕ → ρ
TC Γ (m n)
  | τ is not an arrow := Err
  | otherwise := Check (Split (TC m)) (TC n)
  where
    Split :- (Type : Input) → (Type × Type)
    Split τ → σ := (τ, σ)
    Split ℕ := Err
    Check :- Type → (Type × Type) → Type
    Check (τ1, τ2)σ
      | τ1 = σ := τ2
      | otherwise := Err
    Check τ × σ := Err
    Check ℕ := Err
    τ :- Type
    τ := TC m
-- the following are only in the output language
TC Γ ⟨m; n⟩ = (TC Γ m) × (TC Γ n) -- (pair)
TC Γ max(m, n)
  -- (max)
  | τ = ℕ ∧ σ = ℕ := ℕ
  | otherwise := Err
  where
    τ := TC Γ m
    σ := TC Γ n
TC Γ Vt
  -- (val)
  | τ = btype ℕ := (ℕ → ℕ) → ℕ
  | otherwise := Err
TC Γ Mt
  -- (mod)
  | τ = btype ℕ := (ℕ → ℕ) → ℕ
  | otherwise := Err

```

Code Listing 4.3: Pseudocode for type checking

The type checking function works by using a context to keep track of the types of free variables. This either returns a type or an error. Errors are denoted here as **Err**.

- In the cases of zero, the successor and the recursor, the type can immediately be deduced and then returned.
- In the case of a variable, the type is found by looking up the associated type in the context.
- In the case of an abstraction, the type is deduced by updating the context with the binding and recursing on the body. An arrow is returned, where the domain is the type of the bound variable and the codomain is the type of the body.
- In the case of an application, the type is deduced by recursion on its two components, and checking that the types of the components are compatible. If they are incompatible, then an error is returned, but otherwise, an arrow, where the codomain is the type of the bound variable and the domain is the type of the body.
- In the case of a pair, the type is the product of the types of the two components.
- In the case of maximum, value, modulus, the components are recursed on and the types of the components are compared to specific required types. An error is returned if the type is not as required, otherwise the type is returned as expected.

In Subsection 4.2.1 there is an explanation as to what will happen where one branch encounters an error, and how this will be passed back to the root of the call tree.

The error will contains some information, to indicate what happened but that has been omitted here.

### 4.3.2 Normalisation

Normalisation is the process by which a term is interpreted, and is also necessary to compute moduli of continuity.

A term  $m$  is first type checked, if it does not result in an error, the algorithm proceed.

For a term  $m$  and a context  $\Gamma$ , a set of variable names is computed, from the union of the set of free variable names extracted from the context and a set of bound variables computed from the term itself. This union is denoted  $U(\Gamma, m)$ .

Using a set of fresh variables  $F(\Gamma, m)$ ,

$$F(\Gamma, m) = (\{a, \dots, z\} \cup \{x_i : x \in \{a, \dots, z\}, i \in \mathbb{N}\}) \setminus (U(\Gamma, m)).$$

Each bound variable in  $m$  is matched to a new variable name in  $F(\Gamma, m)$  (sorted lexicographically) according to the “left-to-right” position of the binding in  $m$ . Every abstraction  $\lambda(x : \rho) . n$  is replaced with an abstraction  $\lambda(y : \rho) . n[y/x]$  where  $y$  is a fresh variable that has been matched with  $x$  to produce a new term  $m'$ . This process prevents variable capture, because it makes all variable names unique.

```

Red :- Term → Term
Red (λ(x : ρ) . m y)
  | x = y := m — (2.1), η-reduction
  | otherwise := λ(x : ρ) . (Red (m y)) — note that this does not create a loophole for β-reduction
Red ((λ(x : ρ) . m) n) := m[n/x] — (2.2), β-reduction
Red (((recρ a) f) 0) := a — (2.3)
Red (((recρ a) f) (succ n)) := (f n) (((recρ a) f) n) — (2.4)
Red x := x
Red (λ(x : ρ) . m) = λ(x : ρ) . (Red m)
Red 0 := 0
Red succ := succ
Red recρ := recρ
Red (m n) := (Red m) (Red n)
— the following are only in the output language
Red ⟨Vs; Mt⟩
  | t = s := t — (2.6)
  | otherwise := ⟨Red Vs; Red Mt⟩
Red ⟨m; n⟩ := ⟨Red m; Red n⟩
Red V⟨m;n⟩ := m — (2.6)
Red M⟨m;n⟩ := n — (2.7)
Red Vt := VRed t
Red Mt := MRed t
Red max(succ m, succ n) := succ max(m, n) — (2.8)
Red max(0, n) := n — (2.9)
Red max(m, 0) := m — (2.10)
Red max(m, n)
  | m = n := m — (2.11)
  | otherwise := max(Red m, Red n)

```

Code Listing 4.4: Pseudocode for reductions

Note the order of precedence means that any applicable reduction is checked for before recursion.

Using this inductive translation defined according to

$$m \rightarrow_{\beta} \dots \rightarrow_{\beta} \text{Red } m,$$

a recursive sequence  $m_1, \dots, m_i$  can be obtained according to  $m_{j+1} = \text{Red } m_j$ . Red is the result of some sequence of  $\beta$ -reductions, which may be length zero if it is already in a normal form.

The first term in this sequence is  $m_1 = m'$ , where  $m'$  is the result of the renaming procedure, which is  $\alpha$ -equivalent to  $m$ . The first term in this sequence  $m_j$  such that  $m_{j+1} = m_j$  terminates this sequence, and defines a new translation  $\text{NF } m = m_j$  ( $\text{NF} : \text{Term} \rightarrow \text{Term}$ ), which computes the normal form of  $m$ . The renaming procedure happens a second time, on the normal form.

The way that this is designed means that any rule can be easily encoded, but it is highly inefficient - see Subsection 4.3.4. This sequence is known to be a finite sequence with the last term being the normal form of  $m$ , because it is a sequence of reductions and System  $T$  is a terminating  $\lambda$ -calculus, which makes NF a well-defined translation. (2.2.30 and 2.3.7 in (Troelstra, 1973))

### 4.3.3 Modulus of continuity

The modulus of continuity is fundamental to the project, so its design needs to be clearly stated.

```

btype :- (Type : Input) → Type
btype ℕ = ((ℕ → ℕ) → ℕ) × ((ℕ → ℕ) → ℕ)
btype (τ → σ) = btype τ → btype σ
bcontext :- (Context : Input) → Context
bcontext [x1 : τ1, ..., xn : τn] = [x1b : btype τ1, ..., xnb : btype τn]
ke :- (Type : Input) → (Term : ℕ → btype ρ) → (Term : btype ℕ) → (Term : btype ρ)
ke ℕ g f = ⟨λ(α : ℕ → ℕ) . Vg (Vf α) α; λ(α : ℕ → ℕ) . max(Mg(Vf (α)) (α), Mf (α))⟩
ke (τ → σ) g f = λ(x : btype (σ)) . (λ(k : btype ℕ) . g (k) (x)) (f)
bterm :- Term → Term
bterm x = xb
bterm (m n) = (bterm m) (bterm n)
bterm (λ(x : ρ) . m) = λ(xb : btype (ρ)) . bterm m
bterm 0 = ⟨λ(α : ℕ → ℕ) . 0; λ(α : ℕ → ℕ) . 0⟩
bterm succ = λ(x : ℕb) . ⟨λ(α : ℕ → ℕ) . succ (Vx (α)); Mx⟩
bterm recρ = λ(a : btype ρ) . λ(f : ℕ → btype ρ → btype ρ) . λ(h : btype ℕ) .
    ke ρ (recρb (a) (λ(k : ℕ) . f (λ(α : ℕ → ℕ) . k; λ(α : ℕ → ℕ) . 0))) (h)
ModulusOfContinuity :- (Term : Input, (ℕ → ℕ) → ℕ) → (Term : (ℕ → ℕ) → ℕ)
ModulusOfContinuity f := NF M(bterm f) Ω

```

Code Listing 4.5: Pseudocode for modulus of continuity

The Kleisli extension works as expected according to its definition in Subsection 2.1.5. It is defined here as a translation, rather than as a pure abstraction, because it involves recursion. Note that the type that would be supplied in subscript, is now an argument to the translation. To compute a b-translation, the following operations are performed.

- In the case of a variable, it is renamed by appending b in superscript.
- In the case of an application, the two components are recursed on.
- In the case of an abstraction,
  - the bound variable is renamed by appending b to in superscript,
  - b<sub>term</sub> translates the type of the bound variable,
  - the body is recursed on.
- In the cases of the zero and successor, the relevant term is simply returned.

- In the case of the recursor, a term is returned. Note that unlike in (Xu, 2020), instead of  $ke$  being applied with a single (term) argument, an  $\eta$ -expansion has been used so that there are two (term) arguments that are being applied, so that the Kleisli function can be applied with both arguments, defined here as a translation. This is important in making the function simpler to program.

#### 4.3.4 Examples

It is now necessary to verify that the algorithms explained here work as expected on the examples, as mentioned in Subsection 2.1.6.

##### Type checking

Given the term  $((\text{rec}_{\mathbb{N}} x) (\lambda(u : \mathbb{N}). \text{succ})) 0$ , with  $\Gamma = [x : \mathbb{N}]$ . The type checking pseudocode can be traced as the following.

```

TC [x : ℕ] (((recℕ x) (λ(u : ℕ). succ)) 0)
  TC [x : ℕ] (recℕ x)
    TC [x : ℕ] recℕ = ℕ → (ℕ → ℕ → ℕ) → ℕ → ℕ
    TC [x : ℕ] x = ℕ
    ⇒ TC [x : ℕ] (recℕ x) = (ℕ → ℕ → ℕ) → ℕ → ℕ
  TC [x : ℕ] (λ(u : ℕ). succ))
    TC [u : ℕ, x : ℕ] succ
    ⇒ TC [x : ℕ] (λ(u : ℕ). succ)) = ℕ → ℕ → ℕ
    ⇒ TC [x : ℕ] (λ(u : ℕ). succ) = ℕ → ℕ
  TC [x : ℕ] 0 = ℕ
  ⇒ TC [x : ℕ] ((recℕ x) (λ(u : ℕ). succ)) = ℕ → ℕ
  ⇒ TC [x : ℕ] (((recℕ x) (λ(u : ℕ). succ)) 0) = ℕ

```

Code Listing 4.6: Tracing pseudocode for type checking

This is as expected.

**Normalisation: Addition**

Given the term

$$(\lambda(x : \mathbb{N}). \lambda(y : \mathbb{N}). \text{rec}_{\mathbb{N}}(x) (\lambda(u : \mathbb{N}). \lambda(v : \mathbb{N}). \text{succ}(v)) (y)) (\text{succ}(\text{succ}(\text{succ } 0))) (\text{succ}(\text{succ } 0))$$

the result from the pseudocode can be compared to also produce the same normal form as the reduction proof.

```

Red ((λ(x : ℕ) . λ(y : ℕ) . recℕ(x) (λ(u : ℕ) . λ(v : ℕ) . succ(v)) (y)) (succ(succ(succ 0))) (succ(succ 0)))
= ((λ(y : ℕ) . recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ(v)) (y)) (succ(succ 0)))
Red ((λ(y : ℕ) . recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ(v)) (y)) (succ(succ 0)))
= λ(y : ℕ) . recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ(v)) (succ(succ 0))
Red (λ(y : ℕ) . recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ(v)) (succ(succ 0)))
= recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ(v)) (succ(succ 0))
Red (recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ(v)) (succ(succ 0)))
= recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ(v)) (succ(succ 0))
Red (recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ(v)) (succ(succ 0)))
= (λ(u : ℕ) . λ(v : ℕ) . succ(v)) (succ 0) (recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ(v)) (succ 0))
Red ((λ(u : ℕ) . λ(v : ℕ) . succ(v)) (succ 0) (recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ(v)) (succ 0)))
= (λ(v : ℕ) . succ(v)) (recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ(v)) (succ 0))
Red (succ(recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ(v)) (succ 0)))
= succ(recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ(v)) (succ 0))
Red ((λ(v : ℕ) . succ(v)) (recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ(v)) (succ 0)))
= succ(recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ(v)) (succ 0))
Red (succ(recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ(v)) (succ 0)))
= (Red succ) (Red (recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ(v)) (succ 0)))
= succ((λ(u : ℕ) . λ(v : ℕ) . succ v) (recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ v) (v)) 0)
Red (succ((λ(u : ℕ) . λ(v : ℕ) . succ v) (recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ v) (v)) 0))
= (Red succ) (Red ((λ(u : ℕ) . λ(v : ℕ) . succ v) (recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ v) (v)) 0))
= succ(succ(recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ v) (v)) 0)
= Red (succ(succ(recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ v) (v)) 0))
= (Red succ) (Red (succ(recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ v) (v)) 0))
= succ((Red succ) (Red ((recℕ(succ(succ(succ 0))) (λ(u : ℕ) . λ(v : ℕ) . succ v) (v)) 0)))
= succ(succ(succ(succ(succ 0))))
Red (succ(succ(succ(succ(succ 0)))))
= (Red succ) (Red (succ(succ(succ(succ 0)))))
= succ((Red succ) (Red (succ(succ(succ 0)))))
= succ(succ((Red succ) (Red (succ(succ 0)))))
= succ(succ(succ((Red succ) (Red (succ 0)))))
= succ(succ(succ(succ(succ 0))))

```

Code Listing 4.7: Tracing pseudocode for normalisation (addition)

The reduction strategy that has been employed is very inefficient, but correct. It is relatively simple to program, and is flexible, so rules can be added simply. On the other hand, for complex terms, the reduction sequence produced by Red can be very long.



**Normalisation: Exponentiation**

Given the term the pseudocode can be checked to produce the same normal form.

```

Red ((λ(x : ℕ) . (recℕ (succ 0)) (λ(z : ℕ) . (recℕ 0) (λ(y : ℕ) . (recℕ x) (λ(u : ℕ) . succ)))) (0) (0))
= (recℕ (succ 0)) (λ(z : ℕ) . (recℕ 0) (λ(y : ℕ) . (recℕ 0) (λ(u : ℕ) . succ))) (0)
Red ((recℕ (succ 0)) (λ(z : ℕ) . (recℕ 0) (λ(y : ℕ) . (recℕ 0) (λ(u : ℕ) . succ)))) (0)
= succ 0
Red (succ 0)
= (Red succ) (Red 0)
= succ 0

```

Code Listing 4.8: Tracing pseudocode for normalisation (exponentiation)

The result is as expected. The fact that the length of the trace is relatively short especially compared to the addition example (its pseudocode trace versus its reduction proof) indicates that the complexity likely has more to do with the ‘size’ of the numerals, rather than the size of the overall term.

**Modulus of continuity**

Given the term:

$$t := \lambda(a : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ } (a (a 0)),$$

its modulus of continuity is given by:  $\text{NF } (M_{(\text{b}_{\text{term}} \ t)} \ \Omega)$  meaning the first task to complete, would be to compute the b-translation. Its b-translation can be computed as

```

bterm (λ(a : ℕ → ℕ) . succ (a (a 0)))
= λ(ab : btype(ℕ → ℕ)) . bterm (succ (a (a 0)))
= λ(ab : btype ℕ → btype ℕ) . ((bterm succ) (bterm (a (a 0))))
= λ(ab : btype ℕ → btype ℕ) . ((λ(x : ℕb) . ⟨λ(α : ℕ → ℕ) . succ (Vx(α)); Mx⟩) (ab (bterm (a 0))))
= λ(ab : btype ℕ → btype ℕ) . ((λ(x : ℕb) . ⟨λ(α : ℕ → ℕ) . succ (Vx(α)); Mx⟩) (ab ((bterm a) (bterm 0))))
= λ(ab : btype ℕ → btype ℕ) .
  ((λ(x : btype ℕ) . ⟨λ(α : ℕ → ℕ) . succ (Vx(α)); Mx⟩) (ab (ab (⟨λ(α : ℕ → ℕ) . 0; λ(α : ℕ → ℕ) . 0⟩))))

```

Code Listing 4.9: Tracing pseudocode for b-translation

The trace of the reductions has been omitted because it would likely be too long, but the modulus of continuity would be found as the following.

$$\text{ModulusOfContinuity } (\lambda(a : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ } (a (a 0))) = \dots = \lambda(a : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ } (a 0)$$

# Chapter 5

## Implementation

In this chapter, the setup to use the tool (locally), the core backend script, and the flow of data through the program from being input to being processed to being displayed is discussed. There were four groups of problems to be solved (as identified in Section 4.1):

1. how will the user input the term and context,
2. how will the information be sent to the server and back,
3. how will the term be translated,
4. how will the result be displayed.

### 5.1 Setup

The various versions of software that have been used have been included so that the behaviour can be replicated if necessary.

Table 5.1: Versions of software

Software	Version	Citation
(Mozilla) Firefox	87.0	(Mozilla, 2021)
Node.js	14.16.0	(Node.js Developer Community, 2020)
npm	6.14.11	(npm, 2021)
Glasgow Haskell Compiler	8.6.5	(GHC Team, 2015)
Haskell Stack	1.9.3	(Commercial Haskell SIG , 2018)
Haskell Cabal	2.4.1.0	(Cabal Developer Team, 2018)
Haskell Aeson Package	1.5.6.0	(O'Sullivan, 2021)

Before anything specific is installed, pre-requisites need to be met, in order to make sure that the program would work as expected

The Microsoft Windows operating system and the Mozilla Firefox browser are necessary to use this project. The Windows operating system is necessary because the server script runs a shell command, and shell commands are platform dependent. The Firefox browser is necessary because it is the only major browser on Windows that can represent MathML correctly (MDN Web Docs, 2021c).

### 5.1.1 Installations

The following instructions need be followed to install the core backend script and to run the server script (this would need to happen once to use the tools as many times as necessary).

- Install the version of the Haskell Platform with corresponding Glasgow Haskell Compiler (GHC), version 8.6.5.
- Install Haskell Aeson library, version 1.5.6.0.
- Install Node.js and npm.
- Install the Node.js HTTP package and the Node.js child process packages so that they are node modules contained in the relevant `node_modules` folder (Node.js Developer Community, 2021a) (Node.js Developer Community, 2021b).
- Compile the core Haskell file to an executable (.exe) using `ghc --make core.hs` (GHC Team, 2015).

### 5.1.2 Execution

The following instructions need to be followed to run the tool (this would need to happen every time the tool is opened). After the server script is run once, the webpage can be used as many times as necessary.

- Run the Node.js server script using `node server.js` to run the server.
- Open the main HTML file (`index.html`) in Firefox to run the interface.

## 5.2 Core backend script

The core backend script, written in Haskell, has to be discussed in detail. It is the backbone of the entire project. How it interacts with the interface is discussed separately.

Terms, types, type errors are all encoded as algebraic data types (ADT). This is all necessary because the script needs to be able to manipulate the terms easily. ADTs are suited to this because they can be constructed inductively. Terms can contain other terms, types can contain other types and errors can contain other errors.

Contexts are represented as a wrapper type around a finite map (Leijen and Palamarchuk, 2019).

Types are given by the ADT, `Type`. This is given by the following table.

Table 5.2: Explanation of how types and the corresponding ADT relate

Mathematical construct	ADT construction
$\mathbb{N}$	<code>Nat</code>
$\tau \rightarrow \sigma$	<code>Function tau sigma</code>
$\tau \times \sigma$	<code>Product (tau, sigma)</code>

Formally, this is defined using

```
data Type =
  Product (Type, Type)
| Function Type Type
| Nat
```

Code Listing 5.1: Definition of the type ADT in the core script

Terms are given by the ADT `Term`. This is given by the following table.

Table 5.3: Explanation of how terms and the corresponding ADT relate

Mathematical construct	ADT construction
0	Zero
succ	Succ
$\text{rec}_\rho$	Rec rho
$\lambda(x : \rho). m$	Abstract (x, rho) m
$x$	Variable x
$m\ n$	Apply m n
$V_t$	Value t
$M_t$	Modulus t
$\langle m; n \rangle$	Pair (m, n)
$\max(m, n)$	Max m n

Formally, this is defined using the following code.

```
data Term =
  Variable Var
| Abstract (Var, Type) Term
| Apply Term Term
| Zero
| Succ
| Rec Type
| Pair (Term, Term)
| Max Term Term
| Value Term
| Modulus Term
```

Code Listing 5.2: Definition of the term ADT in the core script

Type errors are given by the ADT `TypeError`. This is given by the following table.

Table 5.4: Explanation of how type errors and the corresponding ADT relate

Description	ADT construction
The variable $x$ was not found in the context	<code>VarNotFoundError message x</code>
The application of $m : \tau$ and $n : \sigma$ is invalid	<code>ApplyError message m n tau sigma</code>
A modulus of continuity was attempted on a term with type $\tau$	<code>ModulusOfContinuityError message tau</code>
A deeper error has occurred (message): $te$	<code>ChainError1 message code te</code>
Two deeper errors have occurred (message): $te1, te2$	<code>ChainError2 message code te1 te2</code>

Formally this is defined using the following code.

```
data TypeError
= VarNotFoundError String Var
| ApplyError String Term Term Type Type
| MaxError String Term Term
| ValueError String Term
| ModulusError String Term
| BTranslationTermError String Term
| BTranslationTypeError String (Type, Type)
| KleisliError String Type
| ModulusOfContinuityError String Type
| ChainError1 String Integer TypeError
| ChainError2 String Integer TypeError TypeError
| ChainErrorMultiple String Integer [TypeError]
```

Code Listing 5.3: Definition of the type error ADT in the core script

Messages are only displayed in the `ChainError1` and `ChainError2` cases (and otherwise omitted), but they have codes, that are omitted. The reason why these are omitted is that these have only been used for the production. There are other constructors for the type `TypeError`, but they are omitted because they are only used in production, and the cases that would cause them are prevented.

Contexts are given by the type `Context`, a wrapper type for the standard strict finite map in Haskell (Leijen and Palamarchuk, 2019).

This is given by the following table.

Table 5.5: Explanation of how contexts and the corresponding ADT relate

Mathematical construct	Type construction
$x_1 : \tau_1, \dots, x_n : \tau_n$	<code>Context (Map.fromList [(x1, tau1), ..., (xn, taun)])</code>

Formally this is defined using the following code.

```
newtype Context = Context (Map.Map Var Type)
```

Code Listing 5.4: Definition of the context type in the core script

Note that `Map.Map` refers to the standard strict finite map data structure in Haskell (Leijen and Palamarchuk, 2019). Contexts are essentially encoded as a map data structure with variable names as the map's keys and types as the map's values. This requires using the `newtype` keyword to provide a wrapper layer to allow for instantiating functions (to represent, parse and so on) specific to this project.

There are other intermediate data structures, such as `InputStructure` and `OutputStructure`, that are used to assist in parsing and encoding, but they are omitted, because they do not serve any purpose in their

own right. Throughout the code, there are various `fix` functions, that add links to a chain of errors, using `ChainError1` and `ChainError2`, and then pass them up to the root of the call tree. There will be references that ignore the `Right` and `Left` constructors that are necessary to use `Either`. Functions here often return `Either TypeError Type` or `Either TypeError Term`, but this will be ignored here because it does not add any new information, and does not correspond to anything from the pseudocode.

### 5.2.1 Type checking

Before any operations can be performed on a term (modulus of continuity, normalisation), it has to be verified as a well-constructed term, with the given context.

The function returns either a `Type`, in the case a valid type is associated with the term, or a `TypeError`, in the case that there is no type associated with the term, because it is invalid.

- In the cases of zero, the successor and the recursor, the type can immediately be deduced and then returned.
- In the case of a variable, the type can be looked up from the `Map` contained within the context. If the variable is not found in the context, then an error is returned.
- In the case of an abstraction, the key-value pair given by the bound variable and its type are inserted into the `Map` contained within the context. If `t` is an error then an error containing it is returned.
- In the case of an application, the two components are type checked and if their types are incompatible, an error is returned, otherwise the codomain of the type of the first component is returned. If the result of type checking either component is an error then an error containing it is returned and if they are both errors then an error containing them both is returned.
- In the case of a pair, the two components are type checked, and the products of the two components are returned. If either one is an error then an error containing it is returned, and if they are both errors then an error containing them both is returned.
- In the cases of a maximum, value, and modulus, the components are type checked and the resulting types are compared to specific required types, where an error is returned if the type is not as required, and otherwise the expected type is returned. If a component results in an error when type checked then an error containing it is returned, and if there are two components that result in an error when type checked then an error containing them both is returned.

Whenever an error occurs in type checking a sub-term, this is fed back to the root of the call tree, using the `ChainError1` and `ChainError2` constructors. Each time, they are used, a 'link' is added to the chain, which will contain a message indicating where the link occurred. This is handled by the various `fix` functions. They create a chain of error messages that the user would be able to use to locate an error in a term.

### 5.2.2 Normalisation

The normalisation function is important, because it is necessary to 'interpret' terms, which is also necessary for the modulus of continuity computation.

To perform a normalisation, the `normalize` function passes the term, where it has been renaming every bound variable with a fresh variable, and the result of a type check with the given context, of the term to the `normalizeHelper` function. If the type check resulted in an error, the `normalizeHelper` function returns that error, but otherwise it passes the term to the `reduceAll` function, which passes the term and the result of the first pass of `reduce` on the term to the `reduceAllHelper` function. The `reduce` function recurses until it finds a reducible expressions and when it reaches one, it returns its reduction. When it reaches a leaf case (a variable, or a constant) it returns the term itself. This function then returns a term, either the same, or with reducible expressions replaced with a reduction - the order of precedence means that this would always happen before recursion. The `reduceAllHelper` function will keep calling `reduce` and comparing the result of each pass until `reduce` returns the same term. This result is returned by the `reduceAllHelper` function, so it is returned by the `reduceAll` function, so it is returned by the `normalizeHelper` function, and therefore it is returned by the `normalize` function.

If the original term failed a type check, the error it resulted in would be returned instead - a term cannot be normalised if it fails a type check.

### 5.2.3 Modulus of continuity

The modulus of continuity is fundamental to the project, so its implementation needs to be clearly stated.

To explain the implementation of modulus of continuity, the various stages have to be explained. A term  $t$  first type checked, if it does not result in an error, then the algorithm proceeds. Then  $M_{t^b\Omega}$  is then normalised. Note that for every free variable  $x : \rho$ , a new variable will be created of the form  $x^b : \rho^b$ , referring to the mapping mentioned ( $\star$ ). Note that bound variables are renamed through normalisation, so this does not occur for bound variables.

- The `bTranslationType` function implements  $b_{\text{type}}$  as mentioned above, either returning a type or an error.
  - On naturals ( $\mathbb{N}$ ), it returns  $((\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}) \times ((\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N})$
  - On arrows, it recurses on the two components as expected. If either component results in an error then an error containing it is returned and if they both result in an error, then an error containing both errors is returned.
  - On products, it returns an error.
- The `bTranslationContext` function implements  $b_{\text{context}}$  as mentioned above, either returning a `Context` (context) or an `TypeError` (error).
  - It returns a new context with every variable renamed by appending  $b$  in superscript and translating every type using the `bTranslationType` function.
  - If any of the types were contained in the context contained products, an error is returned, which contains an array of the errors. Each one corresponds to the type which contained a product in the context.
- The function `ke` returns the Kleisli extension, **ke**, implementing the pseudocode `ke`, either returning a term or an error.
  - The `ke` function passes its term arguments to the `keHelper` function with type checks on them both.
  - In the case that either two type checks are errors, the `keHelper` function returns an error containing it or both in the case that both are errors. Otherwise, it passes its arguments to the `keTypeCheck` function.
  - The `keTypeCheck` function checks that the types of the two arguments are compatible with the definition of the Kleisli extension: that the types of the arguments are  $(\mathbb{N} \rightarrow \rho^b)$  and  $\mathbb{N}^b$ .
  - If it passes this more complex type check then both arguments are passed to the `keHelperHelper` function. Otherwise an error noting that the complex type check was failed is returned - this is done using the `KleisliError`. It also passes a counter, set to 0.
  - Finally `keHelperHelper` recurses on the type argument that was originally passed into `ke` at the very beginning. To avoid the bound variable  $x$  in the definition of the Kleisli extension being captured, every binding of  $x$  is actually numbered with an index (increasing the counter by one on each call), to give  $x_i$ , corresponding to  $x_i$ , where  $i$  indicates the counter. This is fine, because Kleisli is only being used in a leaf case (the recursor), so there does not need to be consideration as to whether  $x$  would capture a different variable. The renaming procedure mentioned above would prevent any other naming issues here that could come up in a reduction.
- The function `bTranslationTerm` implements  $b_{\text{term}}$  (b-translation) as mentioned above, either returning a term or an error.
  - `bTranslationTerm` passes its term argument to `bTranslationTermHelper`. In each case, if a component (type or term) produces an error when translated, an error containing the error or both errors (in the case of an application) will be returned.
    - \* In the case of a variable, it is renamed by appending  $\sim b$ .

- \* In the case of an application, the two components are recursed on. If either component would results in an error, then an error containing it is returned and if they both result in an error, then an error containing them both is returned.
- \* In the case of an abstraction,
  - the bound variable is renamed by appending  $\wedge b$ ,
  - `bTranslationType` translates the type of the bound variable,
  - the body is recursed on, using `bTranslationTermHelper`.

If either `bTranslationType` or `bTranslationTermHelper` return an error then an error containing it is returned, and if they both return errors then an error containing them both is returned.

- \* In the cases of the zero and successor, the relevant term is simply returned.
  - \* In the case of the recursor, a term is returned. Note that unlike in (Xu, 2020), instead of `ke` being applied with a single (term) argument, an  $\eta$ -expansion has been used so that there are two (term) arguments that are being applied, so that the Kleisli function can be applied with both arguments, defined here as a translation. This is important in making the function simpler to program.
  - \* An error will be returned otherwise, because the term is not in the input language.
- The function `modulusOfContinuity` implements `ModulusOfContinuity` (the modulus of continuity) as mentioned above, either returning a term or an error.
    - The function `modulusOfContinuity` passes its arguments and a type check of its term argument to `modulusOfContinuityHelper`. (The fact that a type check has to occur here makes all the error handling in the functions `ke`, `bTranslationTerm`, `bTranslationType`, and `bTranslationContext` unnecessary. This is discussed further in Subsection 7.2.6.)
    - If type check is a type, which does not correspond to  $(\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$ , then an error indicating this is returned.
    - If the type check is an error, then `modulusOfContinuityHelper` returns an error then an error containing it is returned - otherwise the function `normalize` is used to compute the modulus of continuity of the  $M_{s, \Omega}$ , where  $s$  is the b-translation of the original term, unless the b-translation returned an error in which case an error containing it is returned.



### 5.3 Flow of program

How data moves between the user interface (which is set up to be client-side) to the core script (which is set up to be server-side) and back must be explained thoroughly - this has not been explained in previous sections, but it is critical to how the program works.

In short the webpage and the core script communicate using JSON strings. A JSON string is generated by the webpage, which then is sent to the core script, which then parses the string, performs the operation, encoding the result, and sends a string back representing this, which the interface parses and displays. The server script acts as an intermediary subsystem, managing the interaction between the webpage and the core script.

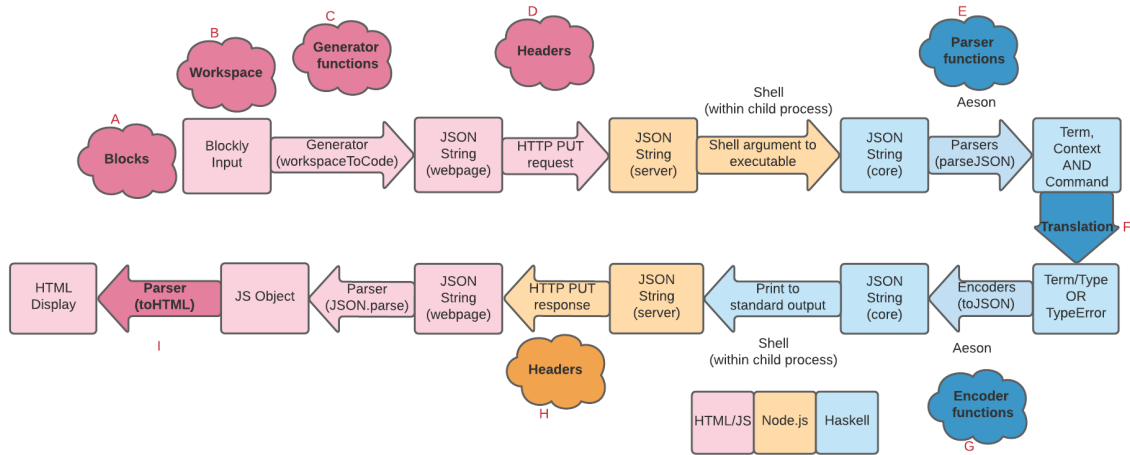


Figure 5.1: Flow of data (note that the letters in the diagram correspond to items and subsections below, clouds indicate that code has been supplied to some other library to produce the relevant function)

1. The user encodes the desired term and context using the blocks that were constructed for this project (A) by placing them into the workspace and manipulating them (B).
2. The user presses the relevant command button between the input and the output spaces.
3. A JSON string consisting of the term, context, and command is constructed using a generator (C).
4. At this point this is where gaps in either the term or the context are checked for, and if there are, a construction error is output and all following items in this list are sidestepped.
5. An HTTP request is made from a client to a server (localhost, port 8000) with this JSON string (D).
6. The server passes the string to the core Haskell script (which has been compiled to an executable) via a shell command in a synchronous child process.
  - (Because the shell removes instances of ", the string that is passed has every occurrence of " replaced by ~, and then within the Haskell program every occurrence of ~ is replaced by ".)
7. The JSON string is converted to a input structure (E).
  - Specific parsers for parsing the input structure, the context, terms, types and were written.
  - These parsers convert an intermediary JSON value (defined by the Aeson library) to an input structure, a context, term, types.
  - These parsers are supplied to the Aeson library, which uses these parses to construct a function that converts a byte string to an input structure (O'Sullivan, 2021).
  - The string that needs to be converted to JSON is converted to byte string and then the function is called on this (Stewart and Coutts, 2021).
8. The input structure is separated into a term, a context and a string corresponding to a command.

9. The function corresponding to the command is executed with the term and context in the input structure. Errors caused here are 'type errors' - construction errors are not type errors (F, Subsection 5.2.1).
  - **Echo** - A term is type checked, and if it successfully passes, the same term will be returned and otherwise, the relevant type error will be returned.
  - **Normal form** - A term is type checked, and if it successfully passes, its type will be returned and otherwise, the relevant type error will be returned.
  - **Modulus of continuity** - A term is type checked, and if it successfully passes, its modulus of continuity will be returned and otherwise, the relevant type error will be returned (on the condition that the term has type  $(\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$ ).
  - **Type check** - A term is type checked, and if it successfully passes, its type will be returned and otherwise, the relevant type error will be returned.
10. The result (either an error or a term or type) is converted to an output structure (G).
  - Specific encoders for encoding the output structure, terms, types and errors were written.
  - These encoders convert the output structure, terms, types, errors to an intermediary JSON value (defined by the Aeson library) (O'Sullivan, 2021).
  - These encoders are supplied to the Aeson library, which uses these encoders to construct a function that converts an output structure to a byte string.
  - This byte string is converted to a string (Stewart and Coutts, 2021).
11. A JSON string consisting of the term, context, and command is constructed.
12. The JSON string is read from the standard output.
  - (Because the shell displays instances of " as \", all instances of \" must be replaced by ".)
13. An HTTP response (whose contents is the JSON string above) is made from the server to the client (H).
14. The JSON string is converted to a JSON object.
15. The JSON object is converted to a string consisting of an HTML element: a MathML element in the case of a term (Ausbrooks et al., 2014), and a nested unordered list in the case of an error (MDN Web Docs, 2021d), where a term within an error will be represented using MathML (I).
16. The output area will contain the HTML element mentioned above by setting the inner HTML of the display area element using the `getElementById` method and the `innerHTML` property (MDN Web Docs, 2021a) (MDN Web Docs, 2021b).

### 5.3.1 Defining Blockly blocks (A)

Blocks are defined, and then supplied to Blockly, which then produces the Blockly interface consisting of a workspace and toolbox based on it.

#### Defining toolbox blocks and the term parent block (using Blockly Developer Tools)

The Blockly Developer Tools provide interfaces for designing blocks, and exporting them so they can be supplied to Blockly, which then constructs the interface based on them (Blockly Developer Team, 2020a).

Two of them are the block factory and the block exporter.

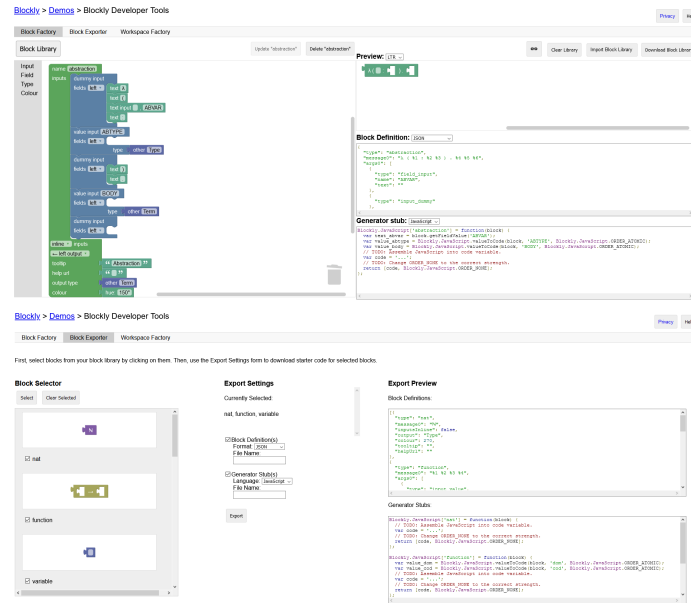


Figure 5.2: Two components of the Blockly Developer Tools (top: Block Factory, bottom: Block Exporter)

- **Block Factory** - an interface itself constructed using Blockly for designing blocks, allowing the user to define the structure (such as which inputs the block takes: values, statements, or some combination) and properties (for example, the help URL and tooltip, whether the inputs are to be inline, how the block will connect to other blocks) of the block.
- **Block Exporter** - an interface for producing definitions of the blocks that can be inserted into a Blockly workspace and stubs of generator functions for converting blocks to code (JavaScript, Python, PHP, Lua, Dart). The JavaScript stubs were used as a starting point for the generating JSON in this project.

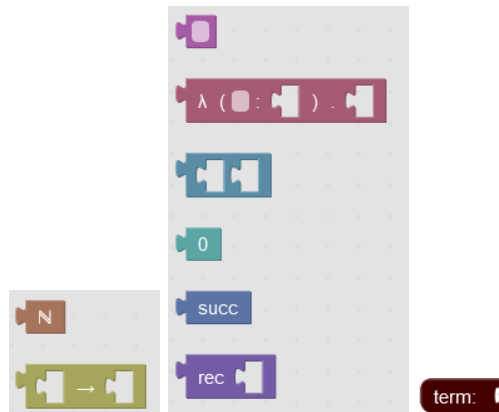


Figure 5.3: The three groups of blocks which were constructed using the Blockly Developer Tools

The definitions that were created using the block factory are listed in Appendix A.3.

### Defining the context block (using a mutator)

For blocks of variable size, more complicated structure, known as a mutator, is necessary.

For example, `list_create` (a standard Blockly block) which creates a list of elements of any type and `field_dropdown` (a block found in the Blockly developer tools) which allows the user to structure a dropdown field (Blockly Developer Team, 2021c).

The 'size' is unknown in these cases, so there needs to be a interface for the user to adjust the size as necessary. This interface is known as a mutator. Mutators can do other things but this will not be discussed (Blockly Developer Team, 2020b).

A mutator works by providing a miniature popup interface so that the user can adjust the main block. This is most commonly arranged as a container block with a workspace that only contains a container and a toolbox that only contains the item block (in this case, the declaration block). The user would drag and drop more declarations into the container block or move blocks out of the container to adjust the size of the main block. Blocks that are not connected to the container would be disabled, and deleted when the mutator interface is closed (by clicking away from it).

In both the `list_create` and `field_dropdown` cases, the 'initial size' is three, which seems like some sort of convention, so that is adopted that here (Blockly Developer Team, 2021c).

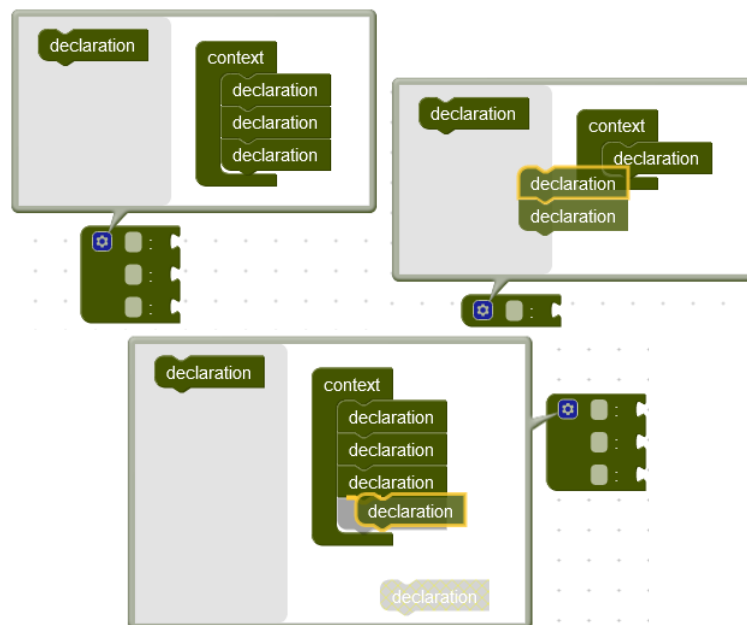


Figure 5.4: Behaviour of the mutator of the context block (top-left: opening the mutator, bottom: inserting an item block, top-right: removing blocks) with the 'size' of the block changing

More complicated techniques to design even more complicated blocks are not listed here.

The definition of the context block is in Appendix A.3.

### 5.3.2 Defining a workspace (using the Blockly Developer Tools) (B)

The workspace is the interface for manipulating blocks - in this case, to construct terms.

The Blockly Developer tools also provide the **Workspace Factory**, an interface for defining a full workspace of simple blocks consisting of toolbox blocks (and how they will be categorised within the toolbox) and initial workspace blocks.

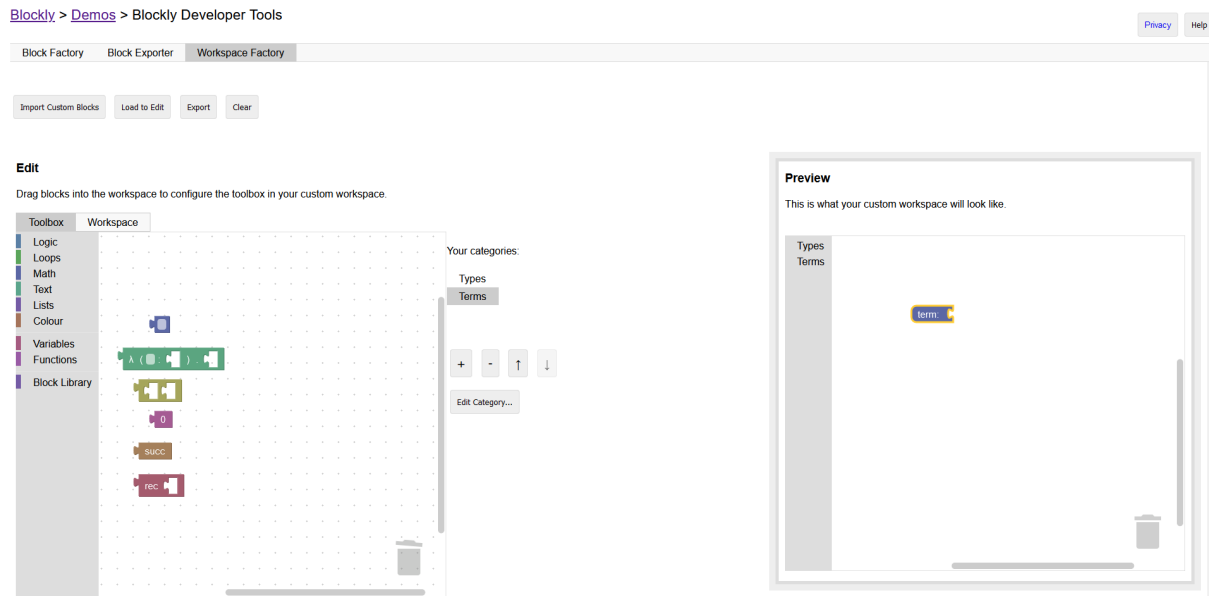


Figure 5.5: Workspace factory

A workspace is defined using various options. There are two initial blocks:

- Context block (`set_context`) - to set the types of free variables,
- Term parent block (`set_term`) - to enable the term being input to be enabled (selected) and all others to be disabled (deselected).

The two initial blocks are specifically set not be deletable, so they cannot be duplicated. 'Orphan blocks' (child blocks, which lack parent blocks) are disabled, so that the only term blocks that are children of the only `set_term` block will be generated, and therefore only its children will be used by the computation. There are two collections (known as 'categories' in the terminology of Blockly - not to be confused with the 'category' property that the JSON strings use to indicate which construction is being used) of objects:

- types,
- terms.

In this the two collections are being used to indicate the 'output type' of the block (Type or Term). The blocks are part of the interface for designing blocks so only those in the input language have corresponding blocks.

### 5.3.3 Defining generators for blocks (C)

Now that the user has clicked a button, the (enabled) blocks are converted to a string - this is done using a generator.

Generator functions have been written to produce JSON strings to be interpreted by the Haskell script. Every block is associated with a generator function by associating each block name with a function within a generator object. The Blockly interface then uses this supplied function to produce a `valueToCode` function. The `workspaceToCode` function is then used get all the blocks as strings.

For technical reasons, a `scrub_` function is also needed.

Terms are generated according to the following table.

Table 5.6: Explanation of how JSON strings are generated for terms

Input	Generated JSON
$\text{rec}_\rho$	<code>{"category": "REC", "rectype": "rho"}</code>
0	<code>{"category": "ZERO"}</code>
$\text{succ}$	<code>{"category": "SUCC"}</code>
$x$	<code>{"category": "VARIABLE", "var": "x"}</code>
$m\ n$	<code>{"category": "APPLY", "left": "m", "right": "n"}</code>
$\lambda(x : \rho) . m$	<pre>{   "category": "ABSTRACT",   "abvar": "x",   "abtype": "rho",   "body": "m" }</pre>

Types are generated according to the following table.

Table 5.7: Explanation of how JSON strings are generated for types

Input	Generated JSON
$\mathbb{N}$	<code>{"category": "NAT"}</code>
$\tau \rightarrow \sigma$	<pre>{   "category": "FUNCTION",   "dom": "tau",   "cod": "sigma" }</pre>

The context written as  $x_1 : \tau_1, \dots, x_n : \tau_n$  would be generated as

```
{
  "declarations": [
    {"vn": x1, "t": tau1},
    ...,
    {"vn": xn, "t": taun}
  ]
}
```

Much of this was done with the assistance of a guide (Blockly Developer Team, 2021a), which explains how to write a generator for the entire JSON specification using blocks mostly belonging to standard library of blocks - this was very helpful especially given that the generator that was written for this project also has JSON output.

For a term  $m$  and a context  $\Gamma$  with command `command`, the final JSON string is constructed as the following.

```
{
  "category": "INPUTSTRUCTURE",
  "term": m,
  "context": gamma,
  "command": command
}
```

#### 5.3.4 Headers for request (D)

Now that the JSON string has been constructed, it needs to be sent to the server by making an HTTP request.

The request is marked as containing being of JSON type. This happens by setting the Content-Type header to `application/json`.

### 5.3.5 Parsers supplied to Aeson (E)

At this point a string has been passed to the backend script, and it needs to be converted back to an ADT to be manipulated by the core script. A string of the form

```
{"category": "INPUTSTRUCTURE", "term": [1], "context": [2], "command": [3]}
```

is converted to an input structure, where [1] is replaced by a term structure, [2] is replaced by a context structure and [3] is replaced by a string corresponding to the command (which indicates the computation being used).

For terms, they are parsed according to the following table.

Table 5.8: Explanation of how terms are parsed from JSON strings

String	Result	Notes
{"category": "ZERO"}	Zero	N/A
{"category": "SUCC"}	Succ	N/A
{"category": "REC", "rectype": [1]}	Rec [1]	[1]: type
{"category": "VARIABLE", "var": [1]}	Variable [1]	[1]: string
{ "category": "APPLY", "left": [1], "right": [2] }	Apply [1] [2]	[1]: term, [2]: term
{ "category": "ABSTRACTION", "abvar": [1], "abtype": [2], "body": [3] }	Abstract ([1], [2]) [3]	[1]: string, [2]: type, [3]: term

For types, they are parsed according to the following table.

Table 5.9: Explanation of how types are parsed from JSON strings

Result	String	Notes
{"category": "NAT"}	Nat	N/A
{ "category": "FUNCTION", "dom": [1], "sigma": [2] }	Function [1] [2]	[1]: type, [2]: type
{ "category": "PRODUCT", "left": [1], "right": [2] }	Product ([1], [2])	[1]: type, [2]: type



Finally, the context JSON string represented by [2] in the input structure above is parsed so that a string of the form

```
{
  "declarations": [
    {"vn": [1, 1], "t": [1, 2]},
    ...,
    {"vn": [n, 1], "t": [n, 2]}
  ]
}
```

is converted to a context, where each of  $[1, 1]$ ,  $\dots$ ,  $[n, 1]$  is replaced by a string and each of  $[1, 2]$ ,  $\dots$ ,  $[n, 2]$  is replaced by a type. This context would be given by the following.

```
Context (Map.fromList [( [1, 1], [1, 2]), ..., ([n, 1], [n, 2])])
```

This is done using the standard strict finite map in Haskell (Leijen and Palamarchuk, 2019).

### 5.3.6 Encoders supplied to Aeson (G)

Now that the result as a value (in the form of an ADT) is obtained, it needs to be converted to a string. The result is either a type, or a term in the case of a non-error.

For types, they are encoded according to the following table.

Table 5.10: Explanation of JSON strings are encoded for types

Result	String
Product (Function (Function Nat Nat) Nat) (Function (Function Nat Nat) Nat)	{"category": "NATB"}
Nat	{"category": "NAT"}
Function [1] [2]	{ "category": "FUNCTION", "dom": [1], "cod": [2] }
Product ([1], [2])	{ "category": "PRODUCT", "left": [1], "right": [2] }

For terms, they are encoded according to the following table.

Table 5.11: Explanation of how JSON strings are encoded for terms

Result	String
Zero	<code>{"category": "ZERO"}</code>
Succ	<code>{"category": "SUCC"}</code>
Rec [1]	<code>{"category": "REC", "rectype": [1]}</code>
Variable [1]	<code>{"category": "VARIABLE", "var": [1]}</code>
Value [1]	<code>{"category": "VALUE", "content": [1]}</code>
Modulus [1]	<code>{"category": "MODULUS", "content": [1]}</code>
Pair ([1], [2])	<code>{   "category": "PAIR",   "left": [1],   "right": [2] }</code>
Apply [1] [2]	<code>{   "category": "APPLY",   "left": [1],   "right": [2] }</code>
Max [1] [2]	<code>{   "category": "MAX",   "left": [1],   "right": [2] }</code>
Abstract ([1], [2]) [3]	<code>{   "category": "ABSTRACT",   "abvar": [1],   "abtype": [2],   "body": [3] }</code>

For type errors, they are encoded according to the following table.

Table 5.12: Explanation of how JSON strings are encoded for type errors

Result	String
VarNotFoundError [1] [2]	{ "category": "VARNOTFOUNDERERROR", "var": [2] }
ApplyError [1] [2] [3] [4]	{ "category": "APPLYERROR", "m": [1], "n": [2], "tau": [3], "sigma": [4] }
ModulusOfContinuityError [1] [2]	{ "category": "MODULUSOFCONTINUITYERROR", "tau": [2] }
ChainError1 [1] [2] [3]	{ "category": "CHAINERROR1", "message": [1], "te": [3] }
ChainError2 [1] [2] [3] [4]	{ "category": "CHAINERROR2", "message": [1], "te1": [3], "te2": [4] }

From this an output structure is constructed. If the result is a type error ([1]), then the output structure is given by the following.

```
{"category": "ERROR", "err"=[1]}
```

If the result is a term or type ([1]), then the output structure is given by the following.

```
{"category": "OKAY", "ok"=[1]}
```

### 5.3.7 Headers for response (H)

Now that a string representing the result has been constructed, it must be sent back to the client. In order to send a response on localhost, the following headers are necessary.

Table 5.13: Headers in response

Name	Value
Content-Type	text/json
Access-Control-Allow-Origin	*
Access-Control-Allow-Methods	GET, POST, OPTIONS, PUT, PATCH, DELETE
Access-Control-Allow-Headers	X-Requested-With,content-type
Access-Control-Allow-Credentials	true

### 5.3.8 Displaying using MathML (I)

At this point, the JSON string will be received by the client, and now the remaining task is to display it. The JSON string will be converted to an HTML string. From that, the inner HTML of a divider is set to the HTML that has been generated.

For types, they are converted according to the following table.

Table 5.14: Explanation of how JSON is converted to MathML for types

JSON	MathML	AT <sub>E</sub> X-equivalent
<code>{"category": "NATB"}</code>	$\begin{array}{c} \text{<msup>} \\ \text{<mi mathvariant="double-struck">} \\ \text{N} \\ \text{</mi>} \\ \text{<mi mathvariant="normal">} \\ \text{b} \\ \text{</mi>} \\ \text{</msup>} \end{array}$	$\mathbb{N}^b$
<code>{"category": "NAT"}</code>	$\begin{array}{c} \text{<mi mathvariant="double-struck">} \\ \text{N} \\ \text{</mi>} \end{array}$	$\mathbb{N}$
<code>{</code> <code>  "category": "FUNCTION",</code> <code>  "dom": tau,</code> <code>  "cod": sigma</code> <code>}</code>	$m \; \&nb\&sp; \&rrarr; \; n$	$\tau \rightarrow \sigma$
<code>{</code> <code>  "category": "PRODUCT",</code> <code>  "left": tau,</code> <code>  "right": sigma</code> <code>}</code>	$\tau \; \&nb\&sp; \&times; \; \sigma$	$\tau \times \sigma$

For terms, they are converted according to the following table.

Table 5.15: Explanation of how JSON is converted to MathML for terms

JSON	MathML	AT <sub>E</sub> X-equivalent
<code>{"category": "ZERO"}</code>	$\langle mn \rangle 0 \langle /mn \rangle$	0
<code>{"category": "SUCC"}</code>	$\langle mi \ mathvariant="normal" \rangle succ \langle /mi \rangle$	succ
<code>{"category": "REC", "rectype": "rho"}</code>	$\langle msub \rangle$ $\langle mi \ mathvariant="normal" \rangle rec \langle /mi \rangle$ $\rho$ $\langle /msub \rangle$	$rec_{\rho}$
<code>{"category": "VARIABLE", "var": "x"}</code>	$\langle mi \rangle x \langle /mi \rangle$	$x$
<code>{   "category": "APPLY",   "left": "m",   "right": "n" }</code>	$m \ \&nbsp; n$	$m \ n$
<code>{   "category": "MAX",   "left": "m",   "right": "n" }</code>	$\langle mi \ mathvariant="normal" \rangle max \langle /mi \rangle$ $\langle mo \rangle ( \langle /mo \rangle$ $m,$ $\&nbsp; n$ $\langle mo \rangle ) \langle /mo \rangle$	$\max(m, n)$
<code>{   "category": "PAIR",   "left": "m",   "right": "n" }</code>	$\&lang \ m; \&nbsp; n \ \&rang;$	$\langle m; n \rangle$
<code>{   "category": "ABSTRACTION",   "abvar": "x",   "abtype": "rho",   "body": "m" }</code>	$\&lambda; (x : \rho)$ $\&nbsp;$ $\langle mo \rangle . \langle /mo \rangle$ $\&nbsp;$ $m$	$\lambda(x : \rho) . m$
<code>{   "category": "VALUE",   "content": "t" }</code>	$\langle msub \rangle$ $\langle mi \ mathvariant="normal" \rangle V \langle /mi \rangle$ $\langle mn \rangle t \langle /mn \rangle$ $\langle /msub \rangle$	$V_t$
<code>{   "category": "MODULUS",   "content": "t" }</code>	$\langle msub \rangle$ $\langle mi \ mathvariant="normal" \rangle M \langle /mi \rangle$ $\langle mn \rangle t \langle /mn \rangle$ $\langle /msub \rangle$	$M_t$

For type errors, they are converted according to the following table.

Table 5.16: Explanation of how JSON is converted to HTML for type errors

JSON	HTML	LaTeX-equivalent
<pre>{   "category":     "VARNOTFOUNDERERROR",   "var":x }</pre>	<pre>&lt;li&gt; var not found in context: &lt;math&gt;&lt;mi&gt;x&lt;/mi&gt;&lt;/math&gt; &lt;/li&gt;</pre>	<ul style="list-style-type: none"> <li>var not found: <math>x</math></li> </ul>
<pre>{   "category": "APPLYERROR",   "m":m,   "n":n,   "tau":tau,   "sigma":sigma }</pre>	<pre>&lt;li&gt; apply error - &lt;math&gt;m&lt;/math&gt;: &lt;math&gt;\tau&lt;/math&gt;: &lt;math&gt;n&lt;/math&gt;: &lt;math&gt;\sigma&lt;/math&gt; &lt;/li&gt;</pre>	<ul style="list-style-type: none"> <li>apply error: <math>m : \tau</math>, <math>n : \sigma</math></li> </ul>
<pre>{   "category":     "MODULUSOFCONTINUITYERROR",   "tau":tau }</pre>	<pre>&lt;ul&gt; modulus of continuity error - &lt;math&gt;\tau&lt;/math&gt; &lt;/ul&gt;</pre>	<ul style="list-style-type: none"> <li>modulus of continuity error: <math>\tau</math></li> </ul>
<pre>{   "category": "CHAINERROR1",   "message":message,   "te":te }</pre>	<pre>&lt;li&gt;message &lt;ul&gt; te &lt;/ul&gt; &lt;/li&gt;</pre>	<ul style="list-style-type: none"> <li>message <math>te</math></li> </ul>
<pre>{   "category": "CHAINERROR2",   "message":message,   "te1":te1,   "te2":te2 }</pre>	<pre>&lt;li&gt;message &lt;ul&gt; te1 te2 &lt;/ul&gt; &lt;/li&gt;</pre>	<ul style="list-style-type: none"> <li>message <math>te_1</math> <math>te_2</math></li> </ul>

Given an output structure, of the form

```
{"category":"OKAY", "ok=ok}"
```

the equivalent HTML would be the following.

```
<math>ok</math>
```

Given an output structure, of the form

```
{"category":"ERROR", "err=err}"
```

the equivalent HTML would be the following.

```
<ul>err</ul>
```

## 5.4 Examples

To verify that the tool works as expected, for the four examples that have been outlined in Subsection 4.3.4 (and by extension, Subsection 2.1.6), they will be input and the output will be compared to the expected output from the tracing pseudocode above.

### Type checking

The term  $((\text{rec}_{\mathbb{N}} x) (\lambda(u : \mathbb{N}). \text{succ})) 0$  would be input as the following.

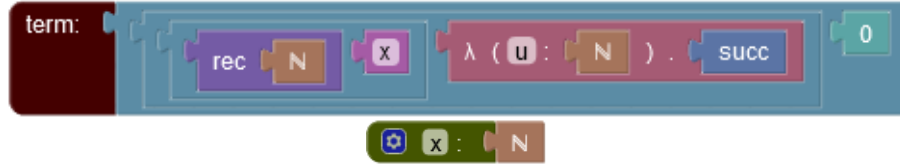


Figure 5.6: Inputting the addition example into the tool

This would be encoded in Haskell as the following.

```
Apply (Apply (Apply (Rec Nat) (Variable "x"))) (Abstract (u, Nat) Succ)) Zero
```

The result would be encoded in Haskell as the following.

```
Nat
```

This would be displayed as the following.

**N**

Figure 5.7: Outputting the result of the addition example from the tool

This is as expected.

### Normalisation: Addition

The addition example would be input as the following.

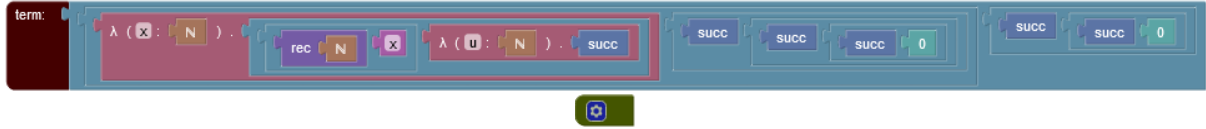


Figure 5.8: Inputting the addition example into the tool

This would be encoded in Haskell as the following.

```
Apply (
  Apply (
    Abstract ("x" , Nat) $ Abstract ("y", Nat) $
    Apply (Apply (Apply (Rec Nat) $ Variable "x")
      (Abstract ("u", Nat) $ Abstract ("v", Nat) $ Apply Succ $ Variable "v"))
    $ Variable "y")
    (Apply Succ (Apply Succ (Apply Succ Zero))))
  (Apply Succ (Apply Succ Zero))
```

The result would be encoded in Haskell as the following.

```
Apply Succ (Apply Succ (Apply Succ (Apply Succ (Apply Succ Zero))))
```

This would be displayed as the following.

**succ (succ (succ (succ (succ 0))))**

Figure 5.9: Outputting the result of the addition example from the tool

This is as expected.



### Normalisation: Exponentiation

The exponentiation example would be input as the following.

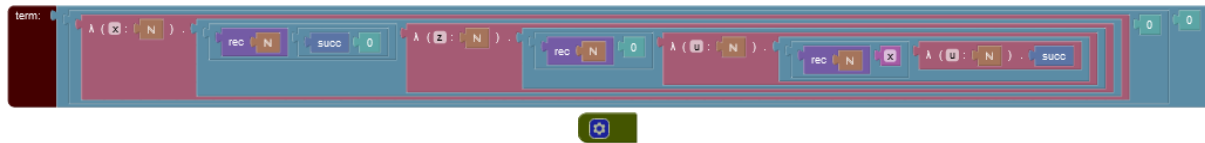


Figure 5.10: Inputting the exponentiation example into the tool

This would be encoded in Haskell as the following.

```
Apply (Apply (Abstract ("x", Nat) Apply (Apply (Rec Nat) (Apply Succ Zero))
  (
    Abstract ("z", Nat) Apply (Apply (Rec Nat) Zero)
    (
      Abstract ("u", Nat) Apply (Apply (Rec Nat) (Variable "x"))
      (
        Abstract ("u", Nat)
        Succ
      )
    )
  )
) Zero) Zero
```

The result would be encoded in Haskell as the following.

```
Apply Succ Zero
```

This would be displayed as the following.

**succ 0**

Figure 5.11: Outputting the result of the exponentiation example from the tool

This is as expected.

### Modulus of continuity

The modulus of continuity example would be input as the following.

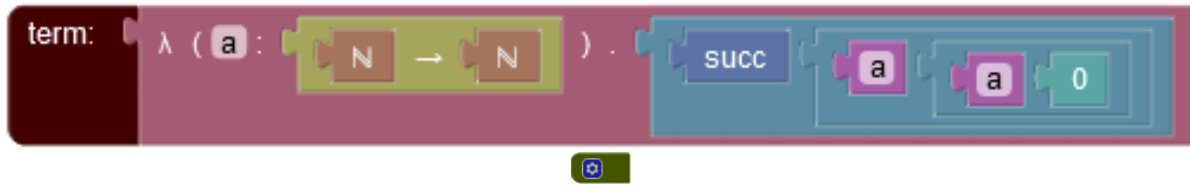


Figure 5.12: Inputting the modulus of continuity example into the tool

This would be encoded in Haskell as the following.

```
Abstract ("a", Function Nat Nat) (Apply Succ (Apply (Variable "a") (Apply (Variable "a") Zero)))
```

The result would be encoded in Haskell as the following.

```
Apply Succ (Apply Succ (Apply Succ (Apply Succ (Apply Succ Zero))))
```

This would be displayed as the following.

$$\lambda(b: \mathbb{N} \rightarrow \mathbb{N}) . \text{succ}(b\ 0)$$

Figure 5.13: Outputting the result of the addition example from the tool

This is as expected.

# Chapter 6

## Testing

The purpose of testing this project is verifying that the computations it produces are as expected.

Screenshots of tests are presented in Appendix B.

Presented below is a key.

There are ordered lists where each item will be referred to as the entity of the list and its number (e.g. Term 1). The two 'B' lists consist of malformed items that will only be used to test for errors. Other items in the main lists may also test for errors. There are unordered lists where each item has a letter and will be referred to as the entity of the list and its letter (e.g. Error C).

### Commands

- Echo (E)
- Normalization (N)
- Modulus of continuity (M)
- Type Check (T)

### Contexts

1. Empty context
2.  $a : \mathbb{N}, b : \mathbb{N} \rightarrow \mathbb{N}, c : (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$

B1.  $\_ : \mathbb{N}$

B2.  $a : \_$

### Terms

1. 0
2. succ
3.  $\text{rec}_{\mathbb{N}}$
4.  $a$
5.  $\lambda(a : \mathbb{N}). 0$
6.  $b a$
7.  $\text{rec}_{\mathbb{N} \rightarrow \mathbb{N}}$
8.  $(\lambda(a : \mathbb{N}). a) 0$
9.  $c$
10.  $\lambda(a : \mathbb{N} \rightarrow \mathbb{N}). 0$
11.  $M_c(\lambda(f : \mathbb{N}^b). \langle \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}). \alpha (V_f(\alpha)); \lambda(\alpha : \mathbb{N} \rightarrow \mathbb{N}). \max(M_f(\alpha), \text{succ}(V_f(\alpha))) \rangle)$  (modulus of continuity of free  $c : (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$ )
12.  $\lambda(x : \mathbb{N}). \lambda(y : \mathbb{N}). \text{rec}_{\mathbb{N}}(x) (\lambda(u : \mathbb{N}). \lambda(v : \mathbb{N}). \text{succ}(v)) (y)$  (the addition operator)
13.  $(\lambda(x : \mathbb{N}). \lambda(y : \mathbb{N}). \text{rec}_{\mathbb{N}}(x) (\lambda(u : \mathbb{N}). \lambda(v : \mathbb{N}). \text{succ}(v)) (y)) (\text{succ}(\text{succ } 0)) (\text{succ } 0)$  (corresponding to  $(2 + a) + (1 + b(0))$ )
14.  $(\lambda(x : \mathbb{N}). \lambda(y : \mathbb{N}). \text{rec}_{\mathbb{N}}(x) (\lambda(u : \mathbb{N}). \lambda(v : \mathbb{N}). \text{succ}(v)) (y)) (\text{succ}(\text{succ } a)) (\text{succ}(b 0))$  (corresponding to 1)

15.  $\lambda(x : \mathbb{N}). (\text{rec}_{\mathbb{N}} x) (\lambda(u : \mathbb{N}). \text{succ})$  (normal form of the addition operator)
16.  $\text{succ} (\text{succ} (\text{succ} 0))$  (corresponding to 3)
17.  $\text{succ} ((\lambda(y : \mathbb{N}). \text{rec}_{\mathbb{N}} (\text{succ} (\text{succ } a)) (\lambda(u : \mathbb{N}). \text{succ})) (b\ 0))$  (corresponding to  $1 + ((2 + a) + b(0))$ )
18.  $\lambda(x : \mathbb{N}). (\text{rec}_{\mathbb{N}} 0) (\lambda(u : \mathbb{N}). (\text{rec}_{\mathbb{N}} x) (\lambda(p : \mathbb{N}). \text{succ}))$  (the multiplication operator)
19.  $(\lambda(x : \mathbb{N}). (\text{rec}_{\mathbb{N}} 0) (\lambda(u : \mathbb{N}). (\text{rec}_{\mathbb{N}} x) (\lambda(p : \mathbb{N}). \text{succ}))) (\text{succ} (\text{succ } 0)) (\text{succ} (\text{succ} (\text{succ } 0)))$  (corresponding to  $2 \cdot 3$ )
20.  $(\lambda(x : \mathbb{N}). (\text{rec}_{\mathbb{N}} 0) (\lambda(u : \mathbb{N}). (\text{rec}_{\mathbb{N}} x) (\lambda(p : \mathbb{N}). \text{succ}))) (\text{succ} (\text{succ} (\text{succ } 0))) (\text{succ} (\text{succ } 0))$  (corresponding to  $3 \cdot 2$ )
21.  $\text{succ} (\text{succ} (\text{succ} (\text{succ} (\text{succ} (\text{succ } 0)))))$  (corresponding to 6)
22.  $\lambda(x : \mathbb{N}). (\text{rec}_{\mathbb{N}} (\text{succ } 0)) (\lambda(z : \mathbb{N}). (\text{rec}_{\mathbb{N}} 0) (\lambda(y : \mathbb{N}). (\text{rec}_{\mathbb{N}} x) (\lambda(u : \mathbb{N}). \text{succ})))$  (the exponentiation operator)
23.  $(\lambda(x : \mathbb{N}). (\text{rec}_{\mathbb{N}} (\text{succ } 0)) (\lambda(z : \mathbb{N}). (\text{rec}_{\mathbb{N}} 0) (\lambda(y : \mathbb{N}). (\text{rec}_{\mathbb{N}} x) (\lambda(u : \mathbb{N}). \text{succ})))) (\text{succ} (\text{succ} (\text{succ } 0))) (\text{succ} (\text{succ } 0))$  (corresponding to  $3^2$ )
24.  $(\lambda(x : \mathbb{N}). (\text{rec}_{\mathbb{N}} (\text{succ } 0)) (\lambda(z : \mathbb{N}). (\text{rec}_{\mathbb{N}} 0) (\lambda(y : \mathbb{N}). (\text{rec}_{\mathbb{N}} x) (\lambda(u : \mathbb{N}). \text{succ})))) (\text{succ} (\text{succ } 0)) (\text{succ} (\text{succ} (\text{succ } 0)))$  (corresponding to  $2^3$ )
25.  $(\lambda(x : \mathbb{N}). (\text{rec}_{\mathbb{N}} (\text{succ } 0)) (\lambda(z : \mathbb{N}). (\text{rec}_{\mathbb{N}} 0) (\lambda(y : \mathbb{N}). (\text{rec}_{\mathbb{N}} x) (\lambda(u : \mathbb{N}). \text{succ})))) (0) (0)$  (corresponding to  $0^0$ )
26.  $\text{succ} (\text{succ} (\text{succ} (\text{succ} (\text{succ} (\text{succ} (\text{succ} (\text{succ} (\text{succ } 0)))))))$  (corresponding to 9)
27.  $\text{succ} (\text{succ} (\text{succ} (\text{succ} (\text{succ} (\text{succ} (\text{succ} (\text{succ } 0))))))$  (corresponding to 8)
28.  $\text{succ } 0$  (corresponding to 1)
29.  $\lambda(a : \mathbb{N} \rightarrow \mathbb{N}). \text{succ } (a\ 0)$  (example 1)
30.  $\lambda(a : \mathbb{N} \rightarrow \mathbb{N}). a\ (a\ (\text{succ } 0))$  (example 2)
31.  $\lambda(a : \mathbb{N} \rightarrow \mathbb{N}). \text{succ} (\text{succ } (a\ (a\ 0)))$  (example 3)
32.  $\lambda(a : \mathbb{N} \rightarrow \mathbb{N}). a\ (a\ (\text{succ } (a\ (a\ 0))))$  (example 4)
33.  $\lambda(a : \mathbb{N} \rightarrow \mathbb{N}). a\ (a\ (\text{succ} (\text{succ } (a\ (a\ (\text{succ} (\text{succ } 0))))))$  (example 5)
34.  $\lambda(a : \mathbb{N} \rightarrow \mathbb{N}). \text{succ } (a\ 0)$  (modulus of continuity of example 1)
35.  $\lambda(a : \mathbb{N} \rightarrow \mathbb{N}). \text{succ } \max(\text{succ } 0, a\ (\text{succ } 0))$  (modulus of continuity of example 2)
36.  $\lambda(a : \mathbb{N} \rightarrow \mathbb{N}). \text{succ } \max(\max(\text{succ } 0, a\ (\text{succ } 0)), a\ (a\ (\text{succ } 0)))$  (modulus of continuity of example 3)
37.  $\lambda(a : \mathbb{N} \rightarrow \mathbb{N}). \text{succ } (a\ (\text{succ } (a\ (\text{succ } (a\ (\text{succ } 0)))))$  (modulus of continuity of example 4)
38.  $\lambda(a : \mathbb{N} \rightarrow \mathbb{N}). \text{succ } \max(\max(a\ 0, \text{succ } (a\ (a\ 0))), a\ (\text{succ } (a\ (a\ 0))))$  (modulus of continuity of example 5)
39.  $d$  (used to cause a Variable Not Found Error, with Context 2)
40.  $(\lambda(d : \mathbb{N}). 0)\ d$  (used to cause a Variable Not Found Error, with Context 2)

B1.  $\text{rec}$ B2.  $\_$  (variable)B3.  $\lambda(\_ : \mathbb{N}). a$ B4.  $\lambda(a : \_). a$ B5.  $\lambda(a : \mathbb{N}). \_$ B6.  $b\ \_$  (application)B7.  $\_ a$  (application)B8.  $\text{rec}_{\mathbb{N} \rightarrow \_}$ B9.  $\text{rec } \_ \rightarrow \mathbb{N}$ B10.  $0\ \text{succ}$ B11.  $\lambda(a : \mathbb{N} \rightarrow \mathbb{N}). \lambda(b : \mathbb{N}). (b\ a)$ **Types**

1.  $\mathbb{N}$
2.  $\mathbb{N} \rightarrow \mathbb{N}$
3.  $(\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$
4.  $\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$
5.  $\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N} \rightarrow \mathbb{N}$
6.  $(\mathbb{N} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N} \rightarrow \mathbb{N})) \rightarrow \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$

**Errors**

- Construction Error (C) - A term or context contains a missing type, term or contains an empty variable name.
- Variable Not Found Error (V) - A variable name is not contained in a context.
- Apply Error (A) - An application does not have a function and argument with compatible types
- Modulus Of Continuity Error (M) - A modulus of continuity was attempted with a term that was not of the type  $(\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$

Only the deepest error in chain errors is important for this classification (there will not be any chains with multiple leaves, because `ChainError2` will be avoided).

**Test classes**

- Normal (N) - Tests where its data will produce a normal result.
- Boundary (B) - Tests where its data represents the furthers the minimum or maximum of what can be input.
- Invalid (I) - Tests where its data is invalid.

Table 6.1: Test plan

Index	Class	Description	Inputs			Expected output	Actual output	Outcome
			Context	Term	Command			
1	N	Simple Zero	1	1	E	Term 1	Term 1	✓
2	N	Simple Zero	1	1	N	Term 1	Term 1	✓
3	N	Simple Zero	1	1	T	Type 1	Type 1	✓
4	N	Simple Successor	1	2	E	Term 2	Term 2	✓
5	N	Simple Successor	1	2	N	Term 2	Term 2	✓
6	N	Simple Successor	1	2	T	Type 2	Type 2	✓
7	N	Simple Recursor and Naturals	1	3	E	Term 3	Term 3	✓
8	N	Simple Recursor and Naturals	1	3	N	Term 3	Term 3	✓
9	N	Simple Recursor and Naturals	1	3	T	Type 5	Type 5	✓
10	N	Simple Variable	2	4	E	Term 4	Term 4	✓
11	N	Simple Variable	2	4	N	Term 4	Term 4	✓
12	N	Simple Variable	2	4	T	Type 1	Type 1	✓
13	N	Simple Abstraction and Naturals	1	5	E	Term 5	Term 5	✓
14	N	Simple Abstraction and Naturals	1	5	N	Term 5	Term 5	✓
15	N	Simple Abstraction and Naturals	1	5	T	Type 2	Type 2	✓
16	N	Simple Application	2	6	E	Term 6	Term 6	✓
17	N	Simple Application	2	6	N	Term 6	Term 6	✓
18	N	Simple Application	2	6	T	Type 1	Type 1	✓
19	N	Simple Recursor and Arrow	1	7	E	Term 7	Term 7	✓
20	N	Simple Recursor and Arrow	1	7	N	Term 7	Term 7	✓
21	N	Simple Recursor and Arrow	1	7	T	Type 6	Type 6	✓
22	N	Simple Normal Form	1	8	N	Term 1	Term 1	✓
23	N	Simplest Modulus of Continuity	2	9	M	Term 11	Term 11	✓
24	N	Simple Modulus of Continuity	1	10	M	Term 10	Term 10	✓
25	N	Normal Form: Addition	1	12	N	Term 16	Term 16	✓
26	N	Normal Form: Addition	1	13	N	Term 17	Term 17	✓
27	N	Normal Form: Addition	1	14	N	Term 18	Term 18	✓
28	N	Normal Form: Multiplication	1	19	N	Term 21	Term 21	✓
29	N	Normal Form: Multiplication	1	20	N	Term 21	Term 21	✓
30	N	Normal Form: Exponentiation	1	23	N	Term 26	Term 26	✓
31	N	Normal Form: Exponentiation	1	24	N	Term 27	Term 27	✓
32	B	Normal Form: Exponentiation	1	25	N	Term 28	Term 28	✓

Continued on next page

Table 6.1 – continued from previous page

Index	Class	Description	Inputs			Expected output	Actual output	Outcome
			Context	Term	Command			
33	N	Modulus of Continuity: November	1	29	M	Term 34	Term 19	✓
34	N	Modulus of Continuity: Example 1	1	30	M	Term 35	Term 36	✓
35	N	Modulus of Continuity: Example 2	1	31	M	Term 36	Term 36	✓
36	N	Modulus of Continuity: Example 3	1	32	M	Term 37	Term 37	✓
37	N	Modulus of Continuity: Example 4	1	33	M	Term 38	Term 38	✓
101	I	Construction Error: Missing Type (Recursor)	1	B1	E	Error C	Error C	✓
102	I	Construction Error: Missing Variable Name (Variable)	2	B2	E	Error C	Error C	✓
103	I	Construction Error: Missing Variable Name (Abstraction)	1	B3	E	Error C	Error C	✓
104	I	Construction Error: Missing Type (Abstraction)	1	B4	E	Error C	Error C	✓
105	I	Construction Error: Missing Term (Abstraction)	1	B5	E	Error C	Error C	✓
106	I	Construction Error: Missing Term (Application: Left)	1	B6	E	Error C	Error C	✓
107	I	Construction Error: Missing Term (Application: Right)	1	B7	E	Error C	Error C	✓
108	I	Construction Error: Missing Type (Arrow: Domain)	1	B8	E	Error C	Error C	✓
109	I	Construction Error: Missing Type (Arrow: Codomain)	1	B9	E	Error C	Error C	✓
110	I	Construction Error: Missing Variable Name (Context)	B1	1	E	Error C	Error C	✓
111	I	Construction Error: Missing Type (Context)	B2	1	E	Error C	Error C	✓
112	I	Variable Name Not Found Error	1	39	E	Error V	Error V	✓
113	I	Application Error	1	B11	E	Error A	Error A	✓
114	I	Application Error	1	B12	E	Error A	Error A	✓
115	I	Variable Name Not Found Error	1	40	E	Error V	Error V	✓
116	I	Modulus Of Continuity Error (Zero)	1	1	M	Error M	Error M	✓

Continued on next page

Table 6.1 – continued from previous page

Index	Class	Description	Inputs			Expected output	Actual output	Outcome
			Context	Term	Command			
117	I	Modulus Of Continuity Error (Successor)	1	2	M	Error M	Error M	✓
118	I	Modulus Of Continuity Error (Recursor, Naturals)	1	3	E	Error M	Error M	✓
119	I	Modulus Of Continuity Error (Variable)	2	4	M	Error M	Error M	✓
120	I	Modulus Of Continuity Error (Abstraction)	1	5	M	Error M	Error M	✓
121	I	Modulus Of Continuity Error (Application)	2	6	M	Error M	Error M	✓
122	I	Modulus Of Continuity Error (Recursor, Arrow)	1	7	M	Error M	Error M	✓



# Chapter 7

## Conclusions

In this chapter, what about the project was successful, what was less so, and how the project will each be discussed in turn.

### 7.1 Successes

I think I am pleased with the state of the project at the date of submission. The core computations of the script (type checking, normalisation, modulus of continuity has been implemented) alongside others. As many of the requirements as possible have been fulfilled.

Scratch was the programming language I first learnt and I had enjoyed learning about System  $T$ , so I have really enjoyed entwining the two in an interesting way (see Chapter 3).

### 7.2 Issues

A major lesson that I learnt from this project is how strange and quirky JavaScript is compared to other programming languages (Mazaika, 2017).

Other issues that were encountered have been discussed below.

#### 7.2.1 Problems with frontend-backend interaction

At the beginning of the development of the project, I did not plan what user interface technology I would be using, I just started writing the backend Haskell script, because I understood that Haskell could communicate with other programming languages, and that the backend script likely needed to be written in a functional programming language.

I did not know that I was going to write the user interface as a website. I only did that because Blockly turned out to be the most convenient system for the Scratch interface that I desired, and the best way to use Blockly is as part of a website. A better process would have been to have a look into what the best user interface technology would have been at the beginning of the programming, and made a decision about which language to write the backend ('core') script in, in a more integrated fashion.

If I were to change this, I would have written the core script in TypeScript instead of Haskell. TypeScript is a 'language which contains all of the features of JavaScript, and an additional layer on top of these: TypeScript's type system' (Microsoft, 2021b) and it is a language which allows for functional programming techniques (Microsoft, 2021a), and this would have made this interaction significantly simpler.

These problems created various issues with hosting (because the backend server would have to run the core script, rather than the frontend client running everything which would have been simpler), and also operating system issues. The shell script that the child process (spawned by the server script) uses to run the executable is dependant on the operating system, and therefore forces the user to use Microsoft Windows (Node.js Developer Community, 2021a).

### 7.2.2 Complication in the code for the b-translation function

At the beginning of writing the Haskell function for b-translation function. (`bTranslationTerm`), I did not realise the important role that type checking would have in the design of the function. By the time I did, I was already mostly finished implementing this part.

At this point I went backwards and wrote the normalisation function `normalize`, which is much simpler, and returned to finish b-translation.

The complications arose because there were issues with the individual cases using functions that might return a `TypeError` and so have a return type of `Either TypeError Term` or `Either TypeError Type`. The case that either it returns `TypeError` is often prevented, so in many places pointless error handling is being used unnecessarily, handling cases that are completely impossible. Because these cases are initially caused at the point of the ‘leaves’ of the function, the code propagates, creating chains of errors that will never occur, ballooning into code that is harder to read.

I ended up having to write an initial type check for the modulus of continuity function (`modulusOfContinuity`) that calls the b-translation function anyway, but it was too late. It would have been better to use an initial type check, and then on you could assume many errors are completely impossible and they would not need to be handled.

### 7.2.3 Problems with Typesetting

The ideal way to represent mathematics on a webpage would be to use MathJax and would convert a  $\text{\LaTeX}$  string to HTML. Unfortunately there were technical issues with doing this dynamically because it has to be done asynchronously (The MathJax Consortium, 2021).

MathML provides an intermediary (but not ideal) option. MathML lacks some of the features of  $\text{\LaTeX}$  which MathJax would properly convert to HTML that would more closely match the equivalent  $\text{\LaTeX}$ . MathJax also outperforms MathML on rendering certain expressions. For instance, the two can be compared, and it is clear that MathJax performs better, particularly in handling nested subscripts and superscripts.

$$\lambda(g : \mathbb{N} \rightarrow \mathbb{N}) . \max(M_{((\text{rec}_{\mathbb{N}b} (\lambda(h : \mathbb{N} \rightarrow \mathbb{N}) . h (\text{succ } 0); \lambda(i : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ } (\text{succ } 0)))} (\lambda(j : \mathbb{N}) . \lambda(k : \mathbb{N}^b) . (\lambda(l : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ } (V_k l); M_k))) (g \ 0) \ g, \text{succ } 0)$$

$$\lambda(g : \mathbb{N} \rightarrow \mathbb{N}) . \max(M_{((\text{rec}_{\mathbb{N}b} (\lambda(h : \mathbb{N} \rightarrow \mathbb{N}) . h (\text{succ } 0); \lambda(i : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ } (\text{succ } 0)))} (\lambda(j : \mathbb{N}) . \lambda(k : \mathbb{N}^b) . (\lambda(l : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ } (V_k l); M_k))) (g \ 0) \ g, \text{succ } 0)$$

Figure 7.1: Typesetting comparison between MathML and MathJax (top: MathJax, bottom: MathML, native  $\text{\LaTeX}$  given in Subsection 7.2.5)

MathML is currently only compatible with Firefox and Safari (MDN Web Docs, 2021c). Together with the requirement to use the Windows operating system, this essentially forces using Firefox to use the tool, because Safari is not available for Windows (Apple, 2020).

### 7.2.4 Problems with Blockly

Blockly is not absolutely suited to a concept like this project in every aspect (although overall, it definitely is). There is no way to preemptively prevent users from using variables that are free variables that do not appear in the context, and preemptively prevent duplicate variable names with different types being present in the context that I found. I think there could be ways to work around this by starting with the standard variable blocks (Blockly Developer Team, 2020c) and limiting the number of blocks which are variable blocks in a workspace (Blockly Developer Team, 2021b), although more work would need to be done to investigate this further.

### 7.2.5 Suboptimal results

For the sake of visual clarity, `add` will denote the addition operator described above:

$$\lambda(x : \mathbb{N}) . (\text{rec}_{\mathbb{N}} x) (\lambda(u : \mathbb{N}) . \text{succ}).$$

It is being laid out this way to show how the tool relies on reductions and cannot simplify based on a term’s meaning.

The way that normalisation works is that there needs to be a specific reduction rule, for the program to make a reduction.

The modulus of continuity of  $\lambda(f : \mathbb{N} \rightarrow \mathbb{N}) . f\ x$ , with the context  $x : \mathbb{N}$ , is  $\lambda(f : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ}\ x$ , but instead, the result from the tool is  $\lambda(d : \mathbb{N} \rightarrow \mathbb{N}) . \max(M_{x^b} d, \text{succ}(V_{x^b} d))$  which is suboptimal.

The modulus of continuity of  $\lambda(f : \mathbb{N} \rightarrow \mathbb{N}) . \text{add}\ (f\ 0)\ (f\ (\text{succ}\ 0))$  (a closed term) is  $\lambda(f : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ}\ (\text{succ}\ 0)$ , but instead the result from the tool is

$$\lambda(g : \mathbb{N} \rightarrow \mathbb{N}) . \max(M_{((\text{rec}_{\mathbb{N}^b} \langle \lambda(h : \mathbb{N} \rightarrow \mathbb{N}) . h\ (\text{succ}\ 0); \lambda(i : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ}\ (\text{succ}\ 0)))} (\lambda(j : \mathbb{N}) . \lambda(k : \mathbb{N}^b) . \langle \lambda(l : \mathbb{N} \rightarrow \mathbb{N}) . \text{succ}\ (V_k\ l); M_k \rangle))\ (g\ 0)\ g, \text{succ}\ 0)),$$

which is further from the optimal result.

The term becomes reaches normal form, but is suboptimal. In order to investigate the limitations, I found a discussion of suboptimal results described in (Hernest, 2007) to be particularly helpful in investigating and discussing suboptimal results of my own tool, but Hernest (2007) uses a different definition of modulus of continuity.

### 7.3 Extensions and Improvements

The project could be extended by using a different definition of the modulus of continuity, or by using MathJax over MathML. The project could also be extended by developing a better interface for the  $\lambda$ -calculus using a visual programming interface using the variable block, while still using Blockly, or even by using more suitable visual programming interface.

### 7.4 Overall conclusions

Overall, the conclusions I drew are that the project is mostly successful. As I had hoped, when I submitted my proposal, the interface is built on a Scratch-like interface, and the core script fully implements moduli of continuity. There is an interface for computing moduli of continuity, which is easy to use. The tool could be improved or extended in various ways, but it fundamentally performs as expected.

# Bibliography

- Alves, S., Fernández, M., Florido, M. and Mackie, I., 2010. Gödel's system  $\mathbf{t}$  revisited. *Theoretical computer science* [Online], 411(11), pp.1484–1500. Available from: <https://doi.org/https://doi.org/10.1016/j.tcs.2009.11.014>.
- Apple, 2020. Update or reinstall Safari for your computer. Publisher: Apple. Available from: <https://support.apple.com/en-gb/HT204416> [Accessed 2021-04-10].
- Ausbrooks, R., Buswell, S., Carlisle, D., Chavchanidze, G., Dalmas, S., Devitt, S., Diaz, A., Dooley, S., Hunter, R., Ion, P., Kohlhase, M., Lazrek, A., Libbrecht, P., Miller, B., Miner, R., Rowley, C., Sargent, M., Smith, B., Soiffer, N., Sutor, R. and Watt, S., 2014. *Mathematical Markup Language (MathML) Version 3.0 2nd Edition* [Online]. (Documentation 2nd edition). World Wide Web Consortium. Publisher: World Wide Web Consortium. Available from: <https://www.w3.org/TR/MathML/> [Accessed 2021-04-28].
- Blockly Developer Team, 2020a. Blockly Developer Tools. Publisher: Google Developers. Available from: <https://developers.google.com/blockly/guides/create-custom-blocks/blockly-developer-tools> [Accessed 2021-04-09].
- Blockly Developer Team, 2020b. Extensions and Mutators | Blockly. Publisher: Google Developers. Available from: <https://developers.google.com/blockly/guides/create-custom-blocks/extensions> [Accessed 2021-04-09].
- Blockly Developer Team, 2020c. Variables | Blockly. Publisher: Google Developers. Available from: <https://developers.google.com/blockly/guides/create-custom-blocks/variables> [Accessed 2021-04-09].
- Blockly Developer Team, 2021a. Build a custom generator. Publisher: Google Developers. Available from: <https://blocklycodelabs.dev/codelabs/custom-generator/index.html?index=..%2F..index#0> [Accessed 2021-04-09].
- Blockly Developer Team, 2021b. Class: Options | Blockly. Publisher: Google Developers. Available from: <https://developers.google.com/blockly/reference/js/Blockly.Options#maxBlocks> [Accessed 2021-04-09].
- Blockly Developer Team, 2021c. google/blockly [Online]. Original-date: 2013-10-25T21:13:33Z. Available from: <https://github.com/google/blockly> [Accessed 2021-04-10].
- Cabal Developer Team, 2018. Cabal: A framework for packaging Haskell software. Publisher: Hackage. Available from: <https://hackage.haskell.org/package/Cabal> [Accessed 2021-04-10].
- Commercial Haskell SIG, 2018. Release v1.9.3 · commercialhaskell/stack. Publisher: GitHub. Available from: [/commercialhaskell/stack/releases/tag/v1.9.3](https://commercialhaskell/stack/releases/tag/v1.9.3) [Accessed 2021-04-10].
- Danner, N., Paykin, J. and Royer, J., 2013. A static cost analysis for a higher-order language [Online]. *7th workshop on programming languages meets program verification*. Available from: <https://doi.org/10.1145/2428116.2428123>.
- GHC Team, 2015. 10.1. Using GHC — Glasgow Haskell Compiler 8.6.5 User's Guide. Publisher: University of Glasgow. Available from: [https://downloads.haskell.org/~ghc/8.6.5/docs/html/users\\_guide/using.html#using-ghc-make](https://downloads.haskell.org/~ghc/8.6.5/docs/html/users_guide/using.html#using-ghc-make) [Accessed 2021-04-09].
- Gödel, V.K., 1958. über eine bisher noch nicht benützte erweiterung des finiten standpunktes. *Dialectica* [Online], 12(3-4), pp.280–287. Available from: <https://doi.org/10.1111/j.1746-8361.1958.tb01464.x>.

- Hernest, M.D., 2007. Synthesis of moduli of uniform continuity by the monotone dialectica interpretation in the proof-system minlog. *Electronic notes in theoretical computer science* [Online], 174(5), pp.141–149. Proceedings of the First International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP 2006). Available from: <https://doi.org/https://doi.org/10.1016/j.entcs.2007.01.023>.
- JSON Schema, 2020. JSON Schema: A Media Type for Describing JSON Documents. Publisher: JSON Schema. Available from: <https://json-schema.org/draft/2020-12/json-schema-core.html> [Accessed 2021-04-27].
- Leijen, D. and Palamarchuk, A., 2019. Data.Map.Strict. Publisher: Hackage. Available from: <https://hackage.haskell.org/package/containers-0.6.4.1/docs/Data-Map-Strict.html> [Accessed 2021-04-17].
- Mazaika, K., 2017. So, what exactly makes JavaScript such a weird programming language? Available from: <http://blog.thefirehoseproject.com/posts/exactly-makes-javascript-weird-programming-language/> [Accessed 2021-04-14].
- MDN Web Docs, 2021a. Document.getElementById() - Web APIs | MDN. Publisher: Mozilla. Available from: <https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementById> [Accessed 2021-04-24].
- MDN Web Docs, 2021b. Element.innerHTML - Web APIs | MDN. Publisher: Mozilla. Available from: <https://developer.mozilla.org/en-US/docs/Web/API/Element/innerHTML> [Accessed 2021-04-24].
- MDN Web Docs, 2021c. <math> - MathML | MDN. Publisher: Mozilla. Available from: <https://developer.mozilla.org/en-US/docs/Web/MathML/Element/math> [Accessed 2021-04-09].
- MDN Web Docs, 2021d. <ul>: The Unordered List element - HTML: HyperText Markup Language | MDN. Publisher: Mozilla. Available from: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/ul> [Accessed 2021-04-28].
- Microsoft, 2021a. Documentation - TypeScript for Functional Programmers. Publisher: Microsoft. Available from: <https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes-func.html> [Accessed 2021-04-09].
- Microsoft, 2021b. Documentation - TypeScript for JavaScript Programmers. Publisher: Microsoft. Available from: <https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html> [Accessed 2021-04-09].
- Mozilla, 2021. Firefox 87.0, See All New Features, Updates and Fixes. Publisher: Mozilla. Available from: <https://www.mozilla.org/en-US/firefox/87.0/releasenotes/> [Accessed 2021-04-09].
- Muchnick, S.S., 1998. *Advanced compiler design and implementation*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.
- Node.js Developer Community, 2020. About this documentation | Node.js v14.16.1 Documentation. Publisher: OpenJS Foundation. Available from: <https://nodejs.org/dist/latest-v14.x/docs/api/documentation.html> [Accessed 2021-04-09].
- Node.js Developer Community, 2021a. Child process | Node.js v14.16.1 Documentation. Publisher: OpenJS Foundation. Available from: [https://nodejs.org/docs/latest-v14.x/api/child\\_process.html#child\\_process\\_child\\_process\\_execsync\\_command\\_options](https://nodejs.org/docs/latest-v14.x/api/child_process.html#child_process_child_process_execsync_command_options) [Accessed 2021-04-09].
- Node.js Developer Community, 2021b. HTTP | Node.js v14.16.1 Documentation. Publisher: OpenJS Foundation. Available from: <https://nodejs.org/docs/latest-v14.x/api/http.html> [Accessed 2021-04-09].
- npm, 2021. npm. Publisher: npm. Available from: <https://www.npmjs.com/package/npm> [Accessed 2021-04-15].
- O'Sullivan, B., 2021. aeson: Fast JSON parsing and encoding. Publisher: Hackage. Available from: <https://hackage.haskell.org/package/aeson> [Accessed 2021-04-09].
- Román, M., 2019. Mikrokosmos: an educational lambda calculus interpreter. *Journal of open source education* [Online], 2(16), p.29. Available from: <https://doi.org/10.21105/jose.00029>.

- Stewart, D. and Coutts, D., 2021. Data.ByteString.Lazy.Internal. Publisher: Hackage. Available from: <https://hackage.haskell.org/package/bytestring-0.11.1.0/docs/Data-ByteString-Lazy-Internal.html> [Accessed 2021-04-28].
- The MathJax Consortium, 2021. MathJax in Dynamic Content — MathJax 3.1 documentation. Revision: 92ce3972. Available from: <http://docs.mathjax.org/en/latest/advanced/typeset.html> [Accessed 2021-04-09].
- Troelstra, A.S., 1973. Models and computability. In: A.S. Troelstra, ed. *Metamathematical investigation of intuitionistic arithmetic and analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp.97–174.
- Wadler, P., Kokke, W. and Siek, J.G., 2020. *Programming language foundations in Agda* [Online]. Available from: <http://plfa.inf.ed.ac.uk/20.07/>.
- Xu, C., 2019. cj-xu/TContinuity [Online]. Original-date: 2019-04-23T13:40:51Z. Available from: <https://github.com/cj-xu/TContinuity> [Accessed 2021-04-10].
- Xu, C., 2020. A syntactic approach to continuity of T-definable functionals. *Logical Methods in Computer Science* [Online], Volume 16, Issue 1. Available from: [https://doi.org/10.23638/LMCS-16\(1:22\)2020](https://doi.org/10.23638/LMCS-16(1:22)2020).

# Appendix A

## Code

The full code is available at: [https://github.bath.ac.uk/km876/LC\\_SystemT](https://github.bath.ac.uk/km876/LC_SystemT).

## A.1 File: core.hs

```
{-# LANGUAGE ExtendedDefaultRules #-}
{-# LANGUAGE OverloadedStrings #-}
{-# LANGUAGE ScopedTypeVariables #-}
{-# OPTIONS_GHC -Wno-deprecated-out-of-scope-variables #-}
{-# OPTIONS_GHC -Wno-deprecated-type-parameters #-}
import qualified Data.ByteString.Lazy.Internal as ByteString (unpackChars, packChars)
import qualified Data.List as List (intercalate)
import qualified Control.Monad as Monad (MonadPlus(mzero))
import qualified Data.Map.Strict as Map (assocs, elems, empty, fromList,
    insert, keysSet, lookup, mapEither, mapKeys, null, union, Map)
import qualified Data.Set as Set (delete, empty, insert, singleton,
    union, Set)
import qualified Data.Vector as Vector (toList)
import qualified System.Environment as SysEnv (getArgs)
import Data.Aeson
    FromJSON(parseJSON), Value(Object), KeyValue((=)), ToJSON(toJSON))
import Data.Aeson.Types (Parser)
import Data.Aeson.Text (encodeToTextBuilder)

----- BASICS -----

replace :: Eq b => b -> b -> [b] -> [b]
replace a b = map $ \c -> if c == a then b else c

main :: IO ()
main = do
    args' <- SysEnv.getArgs
    let arg = head args'
    let arg' = replace '~' (head "\'") arg
    print (commandHandler arg')

-- |Defining variable name
type Var = String

----- Type -----

-- |Defining type
data Type =
    Product (Type, Type)
  | Function Type Type
  | Nat

-- |Equivalence function for type
equivalentType :: Type -> Type -> Bool
equivalentType Nat Nat =
    True
equivalentType (Product (tau1, sigma1)) (Product (tau2, sigma2)) =
    equivalentType tau1 tau2 &&
    equivalentType sigma1 sigma2
equivalentType (Function tau1 sigma1) (Function tau2 sigma2) =
    equivalentType tau1 tau2 &&
    equivalentType sigma1 sigma2
equivalentType _ _ =
    False
```

```
-- |Instantiating said type equivalence (see above)
instance Eq Type where
    (==) = equivalentType

-- |Representation function for types
prettyType :: Type -> String
prettyType Nat = "N"
prettyType (Function tau sigma) = s ++ " -> " ++ t
    where
        s = if sizeType tau > 1 then "(" ++ prettyType tau ++ ")" else prettyType tau
        t = if sizeType sigma > 1 then "(" ++ prettyType sigma ++ ")" else prettyType sigma
prettyType (Product (tau, sigma))
    | tau == natb' && sigma == natb' = "N^b"
    | otherwise = s ++ " <> " ++ t
    where
        s = if sizeType tau > 1 then "(" ++ prettyType tau ++ ")" else prettyType tau
        t = if sizeType sigma > 1 then "(" ++ prettyType sigma ++ ")" else prettyType sigma

-- |Instantiating said type representation (see above)
instance Show Type where
    show = prettyType

instance ToJSON Type where
    toJSON Nat =
        object ["category" .= "NAT"]

    toJSON (Function tau sigma) =
        object ["category" .= "FUNCTION", "dom" .= toJSON
            tau, "cod" .= toJSON sigma]
    toJSON (Product (tau, sigma))
    | tau == natb' && sigma == natb' = object ["category" .= "NATB"]
    | otherwise =
        object ["category" .= "PRODUCT", "left" .= toJSON
            tau, "right" .= toJSON sigma]

parseType :: Value -> Parser Type
parseType (Object o) = do
    c <- o .: "category"
    case c of
        "NAT" -> return Nat;
        "FUNCTION" -> Function <$> o .: "dom" <*> o .: "cod";
        "PRODUCT" -> Product <$> o .: "left" <> o .: "right";
    parseType _ = Monad.mzero

instance FromJSON Type where
    parseJSON = parseType

----- TypeError -----

-- |Defining a type error
data TypeError
    = VarNotFoundError String Var
  | ApplyError String Term Term Type Type
  | MaxError String Term Term
  | ValueError String Term
  | ModulusError String Term
  | BTranslationTermError String Term
  | BTranslationTypeError String (Type, Type)
  | KleisliError String Type
```



```

| ModulusOfContinuityError String Type
| ChainError1 String Integer TypeError
| ChainError2 String Integer TypeError
| ChainErrorMultiple String Integer [TypeError]

-- |Representation function for typeErrors
prettyTypeError :: Int -> TypeError -> String
prettyTypeError i (VarNotFoundError message var) = concat (replicate i "\t" ++
  ["var not found error error: ", var, " (", message, ")"])
prettyTypeError i (ApplyError message m n tau sigma) = concat (replicate i "\t" ++
  ["application error: [", show m, ": ", show tau, "], [", show n, ": ", show sigma, " (",
  message, ")"])
prettyTypeError i (MaxError message m n) = concat (replicate i "\t" ++
  ["max error: ", show m, ", ", show n, " (", message, ")"])
prettyTypeError i (ValueError message m) = concat (replicate i "\t" ++
  ["value error: ", show m, " (", message, ")"])
prettyTypeError i (ModulusError message m) = concat (replicate i "\t" ++
  ["modulus error: ", show m, " (", message, ")"])
prettyTypeError i (BTranslationTermError message m) = concat (replicate i "\t" ++
  ["b-translation (term) error: ", show m, " (", message, ")"])
prettyTypeError i (BTranslationTypeError message (tau, sigma)) = concat (replicate i "\t" ++
  ["b-translation (type) error: ", show tau, ", ", show sigma, " (", message, ")"])
prettyTypeError i (KleisliError message tau) = concat (replicate i "\t" ++
  ["kleisli error: ", show tau, " (", message, ")"])
prettyTypeError i (ModulusOfContinuityError message tau) = concat (replicate i "\t" ++
  ["modulus of continuity error: ", show tau, " (", message, ")"])
prettyTypeError i (ChainError1 message code te) = concat (replicate i "\t" ++
  [message, ": ", show code, "\n", prettyTypeError (i + 1) te])
prettyTypeError i (ChainError2 message code te1 te2) = concat (replicate i "\t" ++
  [message, ": ", show code, "\n", prettyTypeError (i + 1) te1, "\n", prettyTypeError (i + 1)
  te2])
prettyTypeError i (ChainErrorMultiple message code errors) = concat (replicate i "\t" ++
  [message, ": ", show code, "\n"]) ++ unlines (map (prettyTypeError (i + 1)) errors)

-- |Representation function for typeErrors
instance Show TypeError where
  show = prettyTypeError 0

instance ToJSON TypeError where
  toJSON (VarNotFoundError message var) = object ["category" .=
    "VARNOTFOUNDERERROR", "var" .= var] -- can occur using user input
  toJSON (ApplyError message m n tau sigma) = object ["category" .= "APPLYERROR",
    "m" .= toJSON m, "n" .= toJSON n, "tau" .= toJSON tau, "sigma" .= toJSON sigma]
  toJSON (MaxError message m n) = object ["category" .= "MAXERROR",
    "m" .= toJSON m, "n" .= toJSON n] -- can occur using user input
  toJSON (ValueError message m) = object ["category" .= "VALUEERROR",
    "m" .= toJSON m]
  toJSON (ModulusError message m) = object ["category" .=
    "MODULUSERERROR", "m" .= toJSON m]
  toJSON (BTranslationTermError message m) = object ["category" .=
    "BTRANSLATIONTYPEERROR", "m" .= toJSON m]
  toJSON (BTranslationTypeError message (tau, sigma)) = object ["category" .=
    "BTRANSLATIONTERMERROR", "tau" .= toJSON tau, "sigma" .= toJSON sigma]
  toJSON (KleisliError message tau) = object ["category" .=
    "KLEISLIERROR", "tau" .= toJSON tau]

```

```

toJSON (ModulusOfContinuityError message tau) = object ["category" .=
  "MODULUSOFCONTINUITYERROR", "tau" .= toJSON tau] -- can occur using user
input
toJSON (ChainError1 message code te) = object ["category" .=
  "CHAINERROR1", "message" .= message, "te" .= toJSON te] -- can occur using user
input
toJSON (ChainError2 message code te1 te2) = object ["category" .=
  "CHAINERROR2", "message" .= message, "te1" .= toJSON te1, "te2" .= toJSON te2]
-- can occur using user input
toJSON (ChainErrorMultiple message code errors) = object ["category" .=
  "CHAINERRORMANY", "message" .= message, "errors" .= toJSON errors] -- can occur
using user input

```

----- Term -----

-- |Defining term

```

data Term =
  Variable Var
| Abstract (Var, Type) Term
| Apply Term Term
| Zero
| Succ
| Rec Type
| Pair (Term, Term)
| Max Term Term
| Value Term
| Modulus Term

```

-- |Representation function for terms

```

prettyTerm :: Term -> String
prettyTerm Zero = "0"
prettyTerm m
  | isNumeral m = prettyNumeral m
  | otherwise = prettyTermHelper m
where
  prettyTermHelper :: Term -> String
  prettyTermHelper (Variable x) = x
  prettyTermHelper (Abstract (x, rho) m) = "\\(" ++ denote (x, rho) ++ ". " ++ (if sizeTerm
    m > 1 then "(" ++ s ++ ")" else s)
    where s = prettyTerm m
  prettyTermHelper (Apply m n) = s ++ " " ++ t
    where
      s = if sizeTerm m > 1 then "(" ++ prettyTerm m ++ ")" else prettyTerm m
      t = if sizeTerm n > 1 then "(" ++ prettyTerm n ++ ")" else prettyTerm n
  prettyTermHelper Succ = "succ"
  prettyTermHelper Zero = "0"
  prettyTermHelper (Pair (m, n)) = "<" ++ prettyTerm m ++ "; " ++ prettyTerm n
    ++ ">"
  prettyTermHelper (Value m) = if sizeTerm m > 1 then "V_" ++ s ++ "}" else
    "V_" ++ s where s = show m
  prettyTermHelper (Modulus m) = if sizeTerm m > 1 then "M_" ++ s ++ "}" else
    "M_" ++ s where s = show m
  prettyTermHelper (Max m n) = "max(" ++ prettyTerm m ++ ", " ++ prettyTerm n
    ++ ")"
  prettyTermHelper (Rec rho) = "rec[" ++ show rho ++ "]"
prettyNumeral :: Term -> String

```

```

prettyNumeral = prettyNumeralHelper 0
  where
    prettyNumeralHelper :: Integer -> Term -> String
    prettyNumeralHelper n (Apply Succ m) = prettyNumeralHelper (n + 1) m
    prettyNumeralHelper n Zero          = "numeral " ++ show n
    prettyNumeralHelper _ _             = error "won't happen"

-- /Instantiating said term representation (see above)
instance Show Term where
  show = prettyTerm

-- /Equivalence function for term
equivalentTerm :: Term -> Term -> Bool
equivalentTerm (Variable x) (Variable y) = x == y
equivalentTerm (Abstract (x, tau) m) (Abstract (y, sigma) n) = tau == sigma && rename (x, y) m
  == n && rename (y, x) n == m
equivalentTerm (Apply m n) (Apply u v) = m == u && n == v
equivalentTerm Zero Zero = True
equivalentTerm Succ Succ = True
equivalentTerm (Rec tau) (Rec sigma) = tau == sigma
equivalentTerm (Pair (m, n)) (Pair (u, v)) = m == u && n == v
equivalentTerm (Max m n) (Max u v) = m == u && n == v || m == v
  && n == u
equivalentTerm (Value m) (Value n) = m == n
equivalentTerm (Modulus m) (Modulus n) = m == n
equivalentTerm _ _ = False

-- instantiating said type equivalence (see above)
instance Eq Term where
  (==) = equivalentTerm

instance ToJSON Term where
  toJSON Zero = object ["category" .= "ZERO"]
  toJSON Succ = object ["category" .= "SUCC"]
  toJSON (Rec rho) = object ["category" .= "REC", "rectype" .= toJSON rho]
  toJSON (Apply m n) = object ["category" .= "APPLY", "left" .= toJSON m, "right"
    .= toJSON n]
  toJSON (Pair (m, n)) = object ["category" .= "PAIR", "left" .= toJSON m, "right"
    .= toJSON n]
  toJSON (Abstract (x, tau) m) = object ["category" .= "ABSTRACT", "abvar" .= x, "abtype"
    .= toJSON tau, "body" .= toJSON m]
  toJSON (Variable x) = object ["category" .= "VARIABLE", "var" .= x]
  toJSON (Max m n) = object ["category" .= "MAX", "left" .= toJSON m, "right" .=
    toJSON n]
  toJSON (Value m) = object ["category" .= "VALUE", "content" .= toJSON m]
  toJSON (Modulus m) = object ["category" .= "MODULUS", "content" .= toJSON m]

parseTerm :: Value -> Parser Term
parseTerm (Object o) = do
  c <- o .: "category";
  case c of
    "VARIABLE" -> Variable <$> o .: "var";
    "APPLY" -> Apply <$> o .: "left" <*> o .: "right";
    "ZERO" -> return Zero;
    "SUCC" -> return Succ;
    "REC" -> Rec <$> o .: "rectype";

```

```

"PAIR" -> Pair <$> o .: "left" <> o .: "right"
"ABSTRACT" -> fixAbstract <$> o .: "abvar" <*> o .: "abtype" <*> o .: "body"
  where
    fixAbstract :: String -> Type -> Term -> Term
    fixAbstract x tau = Abstract (x, tau)
parseTerm _ = Monad.mzero

instance FromJSON Term where
  parseJSON = parseTerm

----- Context -----

-- /Defining context
newtype Context = Context (Map.Map Var Type)

-- /Denotes a pair of a variable name and type, in the format
denote :: (Var, Type) -> String
denote (m, tau) = m ++ ": " ++ show tau

-- /Denotes the content of a context, with commas (no comma at end)
denoteAll :: [(Var, Type)] -> String
denoteAll = List.intercalate ", " . map denote

prettyContext :: Context -> String
prettyContext (Context gamma)
  | Map.null gamma = "/"
  | otherwise = "{" ++ denoteAll (Map.assocs gamma) ++ "}"

-- /Instantiates said context representation (see above)
instance Show Context where
  show = prettyContext

data Declaration' = Declaration' Var Type
newtype Declaration = Declaration (Var, Type)
newtype DecList = DecList [Declaration]

prettyDecList :: DecList -> String
prettyDecList (DecList decs) = show decs

parseDeclaration :: Value -> Parser Declaration
parseDeclaration o = transformDeclaration <$> parseJSON o
  where
    transformDeclaration :: Declaration' -> Declaration
    transformDeclaration (Declaration' vn t) = Declaration (vn, t)

parseDeclaration' :: Value -> Parser Declaration'
parseDeclaration' (Object o) = Declaration' <$> o .: "vn" <*> o .: "t"
parseDeclaration' _ = Monad.mzero

instance FromJSON Declaration' where
  parseJSON = parseDeclaration'

instance FromJSON Declaration where
  parseJSON = parseDeclaration

instance Show DecList where

```

```

show = prettyDeclList
instance Show Declaration where
show = prettyDeclaration
instance Show Declaration' where
show = prettyDeclaration'

prettyDeclaration' :: Declaration' -> String
prettyDeclaration' (Declaration' vn t) = vn ++ ": " ++ show t

prettyDeclaration :: Declaration -> String
prettyDeclaration (Declaration (vn, t)) = vn ++ ": " ++ show t

instance FromJSON DecList where
parseJSON (Object o) = do
pts <- o .: "declarations"
ptsList <- mapM parseJSON $ Vector.toList pts
return $ DecList ptsList
parseJSON _ = Monad.mzero

parseDeclList :: Value -> Parser DecList
parseDeclList (Object o) = do
pts <- o .: "declarations"
DecList <$> parseJSON pts
parseDeclList _ = Monad.mzero

parseContext :: Value -> Parser Context
parseContext v = Context . Map.fromList . map unpackDeclaration . (\(DecList dl) -> dl) <$>
parseDeclList v
where
unpackDeclaration :: Declaration -> (Var, Type)
unpackDeclaration (Declaration (vn, t)) = (vn, t)

instance FromJSON Context where
parseJSON = parseContext

----- Input Structure -----

data InputStructure = InputStructure String Context Term

parseInputStructure :: Value -> Parser InputStructure
parseInputStructure (Object o) = InputStructure <$> o .: "command" <*> o .: "context" <*> o
.: "term"
parseInputStructure _ = Monad.mzero

instance FromJSON InputStructure where
parseJSON = parseInputStructure

-- /Representation function for types
prettyInputStructure :: InputStructure -> String
prettyInputStructure (InputStructure comand context term) = show context ++ "\n" ++ show
term

instance Show InputStructure where
show = prettyInputStructure

----- Output Structure -----

```

```

data OutputStructure = Ok (Either Type Term) | Err TypeError

```

```

instance ToJSON OutputStructure where
toJSON (Ok (Right term)) = object ["category" .= "OKAY", "ok" .= term]
toJSON (Ok (Left tau)) = object ["category" .= "OKAY", "ok" .= tau]
toJSON (Err te) = object ["category" .= "ERROR", "err" .= te]

```

```

-- /Representation function for types
prettyOutputStructure :: OutputStructure -> String
prettyOutputStructure (Ok (Right term)) = show term
prettyOutputStructure (Ok (Left tau)) = show tau
prettyOutputStructure (Err te) = show te

```

```

instance Show OutputStructure where
show = prettyOutputStructure

```

```

----- ADDITIONAL FUNCTIONS -----

```

```

-- /Splits an type into its curried domain and codomain
splitType :: Type -> Maybe (Type, Type)
splitType (Function tau sigma) = Just (tau, sigma)
splitType _ = Nothing

-- /Finds the type of a term given its context
{-handles bound variables by inserting the (variable name, type) pair in the context
overwriting any pair (variable name, type') pair, so different branches can proceed independently
-}
typeCheck :: Context -> Term -> Either TypeError Type
typeCheck (Context gamma) (Variable x) = fix ft
where
fix :: Maybe Type -> Either TypeError Type
fix Nothing = Left (VarNotFoundError "variable missing from context" x)
fix (Just rho) = Right rho
ft :: Maybe Type
ft = Map.lookup x gamma
typeCheck (Context gamma) (Abstract (x, tau) m) = fix tc
where
tc :: Either TypeError Type
tc = typeCheck (Context (Map.insert x tau gamma)) m
fix :: Either TypeError Type -> Either TypeError Type
fix (Right sigma) = Right (Function tau sigma)
fix (Left te) = Left (ChainError1 "occurred in type check, abstraction" 2 te)
typeCheck context (Pair (m, n)) = fix tau' sigma'
where
tau' :: Either TypeError Type
tau' = typeCheck context m
sigma' :: Either TypeError Type
sigma' = typeCheck context n
fix :: Either TypeError Type -> Either TypeError Type -> Either TypeError Type
fix (Right tau'') (Right sigma'') = Right (Product (tau'', sigma''))
fix (Right _) (Left te) = Left (ChainError1 "occurred in type check, pair / right"
3 te)
fix (Left te) (Right _) = Left (ChainError1 "occurred in type check, pair / left" 4
te)

```

```

    fix (Left te1) (Left te2) =      Left (ChainError2 "occurred in type check, pair / both" 5
      te1 te2)
typeCheck context (Max m n) = fix tc1 tc2
  where
    tc1 :: Either TypeError Type
    tc1 = typeCheck context m
    tc2 :: Either TypeError Type
    tc2 = typeCheck context n
    fix :: Either TypeError Type -> Either TypeError Type -> Either TypeError Type
    fix (Right Nat) (Right Nat) = Right Nat
    fix (Right _) (Right _) = Left (MaxError "occurred in type check, max" m n)
    fix (Left te1) (Right _) = Left (ChainError1 "occurred in type check, max / left" 6 te1)
    fix (Right _) (Left te2) = Left (ChainError1 "occurred in type check, max / right" 7 te2)
    fix (Left te1) (Left te2) = Left (ChainError2 "occurred in type check, max / both" 8 te1
      te2)
typeCheck context (Apply m n) = fix2 tc1 tc2
  where
    tc1 = typeCheck context m
    tc2 = typeCheck context n
    fix1 :: Type -> Type -> Maybe (Type, Type) -> Either TypeError Type
    fix1 tau' sigma' Nothing = Left (ApplyError "occurred in type check, apply" m n tau'
      sigma')
    fix1 tau' sigma' (Just (tau, sigma))
      | sigma' == tau = Right sigma
      | otherwise = Left (ApplyError "occurred in type check, apply" m n tau' sigma')
    fix2 :: Either TypeError Type -> Either TypeError Type -> Either TypeError Type
    fix2 (Right tau) (Right sigma) = fix1 tau sigma (splitType tau)
    fix2 (Left te1) (Right _) = Left (ChainError1 "occurred in type check, apply / left" 9
      te1)
    fix2 (Right _) (Left te2) = Left (ChainError1 "occurred in type check, apply / right"
      10 te2)
    fix2 (Left te1) (Left te2) = Left (ChainError2 "occurred in type check, apply / both"
      11 te1 te2)
typeCheck _ Succ = Right (Function Nat Nat)
typeCheck _ Zero = Right Nat
typeCheck _ (Rec rho) = Right (Function rho (Function (Function Nat (Function rho rho)) (Function
  Nat rho)))
typeCheck context (Value m) = fix tc
  where
    tc :: Either TypeError Type
    tc = typeCheck context m
    fix :: Either TypeError Type -> Either TypeError Type
    fix (Right tau)
      | tau == natb = Right natb'
      | otherwise = Left (ValueError "occurred in type check, value" m)
    fix (Left te) = Left (ChainError1 "occurred in type check, value" 12 te)
typeCheck context (Modulus m) = fix tc
  where
    tc :: Either TypeError Type
    tc = typeCheck context m
    fix :: Either TypeError Type -> Either TypeError Type
    fix (Right tau)
      | tau == natb = Right natb'
      | otherwise = Left (ModulusError "occurred in type check, modulus" m)
    fix (Left te) = Left (ChainError1 "occurred in type check, modulus" 13 te)

```

```

----- USEFUL FUNCTIONS -----

-- /Whether or not a term can be produced
isNumeral :: Term -> Bool
isNumeral (Apply Succ m) = isNumeral m
isNumeral Zero = True
isNumeral _ = False

-- /Computes the number of symbols in a type
sizeType :: Type -> Int
sizeType Nat = 1
sizeType (Function _ _) = 2
sizeType (Product (tau, sigma))
  | tau == natb' && sigma == natb' = 1
  | otherwise = 2
sizeTerm :: Term -> Int
sizeTerm m
  | isNumeral m = 1
  | otherwise = sizeTermHelper m

-- /Computes the number of symbols in a term
sizeTermHelper :: Term -> Int
sizeTermHelper (Apply _ _) = 2
sizeTermHelper (Abstract _ _) = 3
sizeTermHelper (Value _) = 2
sizeTermHelper (Modulus _) = 2
sizeTermHelper _ = 1

-- /Computes a list of free variable names of a term
freeNames :: Term -> Set.Set Var
freeNames (Variable x) = Set.singleton x
freeNames (Apply m n) = Set.union (freeNames m) (freeNames n)
freeNames (Pair (m, n)) = Set.union (freeNames m) (freeNames n)
freeNames (Max m n) = Set.union (freeNames m) (freeNames n)
freeNames (Abstract (x, _) m) = Set.delete x (freeNames m)
freeNames (Value m) = freeNames m
freeNames (Modulus m) = freeNames m
freeNames _ = Set.empty

-- /Computes the number of bound variables of a term
boundNum :: Term -> Int
boundNum (Apply m n) = boundNum m + boundNum n
boundNum (Abstract (_, _) m) = 1 + boundNum m
boundNum (Pair (m, n)) = boundNum m + boundNum n
boundNum (Max m n) = boundNum m + boundNum n
boundNum (Value m) = boundNum m
boundNum (Modulus m) = boundNum m
boundNum _ = 0

-- /Computes a list of bound variable names of a term
boundNames :: Term -> Set.Set Var
boundNames (Apply m n) = Set.union (boundNames m) (boundNames n)
boundNames (Abstract (x, _) m) = Set.insert x (boundNames m)
boundNames (Pair (m, n)) = Set.union (boundNames m) (boundNames n)
boundNames (Max m n) = Set.union (boundNames m) (boundNames n)
boundNames (Value m) = boundNames m

```

```

boundNames (Modulus m) = boundNames m
boundNames _ = Set.empty

-- /Renames occurrences of x to y in a term
rename :: (Var, Var) -> Term -> Term
rename (x, y) (Variable z)
  | x == z = Variable y
  | otherwise = Variable z
rename (x, y) (Abstract (z, tau) m)
  | x == z = Abstract (z, tau) m
  | otherwise = Abstract (z, tau) (rename (x, y) m)
rename (x, y) (Apply m n) = Apply (rename (x, y) m) (rename (x, y) n)
rename (x, y) (Max m n) = Max (rename (x, y) m) (rename (x, y) n)
rename (x, y) (Pair (m, n)) = Pair (rename (x, y) m, rename (x, y) n)
rename (x, y) (Value m) = Value (rename (x, y) m)
rename (x, y) (Modulus m) = Modulus (rename (x, y) m)
rename (_, _) m = m

-- /Substitutes m for x in a
substitute :: (Var, Term) -> Term -> Term
substitute (x, m) (Variable y)
  | x == y = m
  | otherwise = Variable y
substitute (x, m) (Abstract (z, tau) n)
  | x == z = Abstract (z, tau) n
  | otherwise = Abstract (z, tau) (substitute (x, m) n)
substitute (x, l) (Apply m n) = Apply (substitute (x, l) m) (substitute (x, l) n)
substitute (x, l) (Pair (m, n)) = Pair (substitute (x, l) m, substitute (x, l) n)
substitute (x, l) (Max m n) = Max (substitute (x, l) m) (substitute (x, l) n)
substitute (x, m) (Value n) = Value (substitute (x, m) n)
substitute (x, m) (Modulus n) = Modulus (substitute (x, m) n)
substitute (_, _) m = m

-- /List of all standard names
allNames :: [Var]
allNames = map (:[]) ['a'..'z'] ++ [x : show i | i <- [0..], x <- ['a'..'z']]
where
  o :: Integer
  o = 1

-- /Computes a set of all names in a context, term
usedNames :: Context -> Term -> Set.Set Var
usedNames (Context gamma) m = Set.union (Map.keysSet gamma) (boundNames m)

-- /Filters used names from a infinite list of standard names
freshNames :: Context -> Term -> [Var]
freshNames context m = filter (notElem usedNames context m) allNames

-- /Helper function to reassign names
reassign :: Context -> Term -> Term
reassign context m' = reassignHelper (freshNames context m') m'
where
  reassignHelper :: [Var] -> Term -> Term
  reassignHelper (x:xs) (Abstract (y, tau) m) = Abstract (x, tau) (reassignHelper xs (rename (y, x) m))

```

```

reassignHelper xs (Apply m n) = Apply (reassignHelper xs m) (reassignHelper (drop
  (boundNum m) xs) n)
reassignHelper xs (Pair (m, n)) = Pair (reassignHelper xs m, reassignHelper (drop
  (boundNum m) xs) n)
reassignHelper xs (Max m n) = Max (reassignHelper xs m) (reassignHelper (drop (boundNum m)
  xs) n)
reassignHelper xs (Value m) = Value (reassignHelper xs m)
reassignHelper xs (Modulus m) = Modulus (reassignHelper xs m)
reassignHelper [] _ = error "won't happen"
reassignHelper _ m = m

```

---

### REDUCTION

---

```

-- /Normalizes a term
normalize :: Context -> Term -> Either TypeError Term
normalize context m = normalizeHelper (typeCheck context m) (reassign context m)
where
  normalizeHelper :: Either TypeError Term -> Term -> Either TypeError Term
  normalizeHelper (Left te) _ = Left te
  normalizeHelper (Right _) m' = Right (reassign context (reduceAll m'))
  where
    reduceAll :: Term -> Term
    reduceAll m = reduceAllHelper m (reduce m)
    where
      reduceAllHelper :: Term -> Term -> Term
      reduceAllHelper prev curr
        | prev == curr = curr
        | otherwise = reduceAllHelper curr (reduce curr)
    reduce :: Term -> Term
    reduce (Abstract (v, tau) (Apply w (Variable v'))))
      | v' == v = w -- eta reduction
      | otherwise = Abstract (v, tau) (reduce (Apply w (Variable v'))))
    reduce (Apply (Abstract (x, _) l) m) = substitute (x, m) l -- beta reduction
    reduce (Apply (Apply (Apply (Rec rho) a) f) (Apply Succ n)) = Apply (Apply f n)
      (Apply (Apply (Apply (Rec rho) a) f) n)
    reduce (Apply (Apply (Apply (Rec _) a) _) Zero) = a
    reduce (Max (Apply Succ m) (Apply Succ n)) = Apply Succ (Max m n)
    reduce (Value (Pair (m, _))) = m
    reduce (Modulus (Pair (_, n))) = n
    reduce (Max m Zero) = m
    reduce (Max Zero n) = n
    reduce (Max m n)
      | m == n = m
      | otherwise = Max (reduce m) (reduce n)
    reduce (Apply m n) = Apply (reduce m) (reduce n)
    reduce (Abstract (y, tau) m) = Abstract (y, tau) (reduce m)
    reduce (Pair (Value s, Modulus t))
      | s == t = s
      | otherwise = Pair (reduce (Value s), reduce (Modulus t))
    reduce (Pair (m, n)) = Pair (reduce m, reduce n)
    reduce (Value m) = Value (reduce m)
    reduce (Modulus m) = Modulus (reduce m)
    reduce m = m

```

```

structureTermOutput :: Either TypeError Term -> OutputStructure
structureTermOutput (Right m) = Ok (Right m)
structureTermOutput (Left te) = Err te

structureTypeOutput :: Either TypeError Type -> OutputStructure
structureTypeOutput (Right m) = Ok (Left m)
structureTypeOutput (Left te) = Err te

commandContainer :: InputStructure -> OutputStructure
commandContainer (InputStructure "E" context term) = echoHandler (typeCheck context term) term
  where
    echoHandler :: Either TypeError Type -> Term -> OutputStructure
    echoHandler (Right _) term = Ok (Right term)
    echoHandler (Left te) _ = Err te
commandContainer (InputStructure "N" context term) = structureTermOutput $ normalize context term
commandContainer (InputStructure "B" context term) = structureTermOutput $
  bTranslationTermNormal context term
commandContainer (InputStructure "O" context term) = structureTermOutput $
  bTranslationTermOmega context term
commandContainer (InputStructure "M" context term) = structureTermOutput $ modulusOfContinuity
  context term
commandContainer (InputStructure "T" context term) = structureTypeOutput $ typeCheck context
  term
commandContainer InputStructure {} = error "won't happen"

decode'' :: FromJSON a => String -> Maybe a
decode'' = decode . ByteString.packChars

encode'' :: ToJSON a => a -> String
encode'' = ByteString.unpackChars . encode

commandHandler :: String -> String
commandHandler inputText = process decoded
  where
    decoded :: Maybe InputStructure
    decoded = (decode'' :: String -> Maybe InputStructure) inputText
    process :: Maybe InputStructure -> String
    process Nothing =
      "{ \"category\": \"ERROR\", \"err\": { \"category\": \"CONTAINERERROR\" } }"
    process (Just a) = encode'' (commandContainer a)

----- b-TRANSLATION -----

-- |Determines whether a type is encodable
encodableType :: Type -> Bool
encodableType Nat = True
encodableType (Function tau sigma) = encodableType tau && encodableType sigma
encodableType (Product (_, _)) = False

-- |Determines whether a term is encodable
encodableTerm :: Term -> Bool
encodableTerm (Abstract (_, tau) m) = encodableTerm m && encodableType tau
encodableTerm (Apply m n) = encodableTerm m && encodableTerm n

```

```

encodableTerm (Variable _) = True
encodableTerm Zero = True
encodableTerm Succ = True
encodableTerm (Rec rho) = encodableType rho
encodableTerm _ = False

numeral :: Integer -> Term
numeral n
  | n > 0 = Apply Succ (numeral (n - 1))
  | otherwise = Zero

-- |Computes the maximum of two terms
max' :: Context -> Term -> Term -> Either TypeError Term
max' context m' n' = fix tau sigma
  where
    tau :: Either TypeError Type
    tau = typeCheck context m'
    sigma :: Either TypeError Type
    sigma = typeCheck context n'
    fix (Right _) (Left te) = Left (ChainError1 "occured in max / right" 14 te)
    fix (Left te) (Right _) = Left (ChainError1 "occured in max / left" 15 te)
    fix (Left te1) (Left te2) = Left (ChainError2 "occured in max / both" 16 te1 te2)
    fix (Right Nat) (Right Nat) = Right (maxHelper m' n')
    where
      maxHelper :: Term -> Term -> Term
      maxHelper (Apply Succ m) (Apply Succ n) = Apply Succ (maxHelper m n)
      maxHelper Zero n = n
      maxHelper m Zero = m
      maxHelper m n
        | m == n = m
        | otherwise = Max m n
    fix (Right _) (Right _) = Left (MaxError "occured in max / incorrect type(s)" m' n')

-- |x ~> x^b
bTranslationVar :: Var -> Var
bTranslationVar x = x ++ "^b"

-- |Single component of N^b
natb' :: Type
natb' = Function (Function Nat Nat) Nat

-- |N^b
natb :: Type
natb = Product (natb', natb')

-- |Unpacks a result assuming it is not an error type
unpack :: Either a b -> b
unpack (Right r') = r'
unpack (Left _) = error "unpack error"

-- |rho ~> rho^b
bTranslationType :: Type -> Either TypeError Type
bTranslationType Nat = Right natb
bTranslationType (Function sigma tau) = fix sigma' tau'

```

```

where
  sigma' :: Either TypeError Type
  sigma' = bTranslationType sigma
  tau' :: Either TypeError Type
  tau' = bTranslationType tau
  fix :: Either TypeError Type -> Either TypeError Type -> Either TypeError Type
  fix (Right sigma'') (Right tau'') = Right (Function sigma'' tau'')
  fix (Right _) (Left te) = Left (ChainError1 "occurred in b-translation (type) / right" 17 te)
  fix (Left te) (Right _) = Left (ChainError1 "occurred in b-translation (type) / left" 18 te)
  fix (Left te1) (Left te2) = Left (ChainError2 "occurred in b-translation (type) / both" 19 te1 te2)
bTranslationType (Product (sigma, tau)) = Left (BTranslationTypeError "occured in b-translation for type" (sigma, tau))

-- /Performs b-translation on every (variable name, type) pair in a context
bTranslationContext :: Context -> Either TypeError Context
bTranslationContext (Context gamma)
  | Map.null errors = Right (Context (Map.mapKeys bTranslationVar types))
  | otherwise = Left (ChainErrorMultiple "occurred in b-translation context" 100 (Map.elems errors))
where
  errors :: Map.Map Var TypeError
  types :: Map.Map Var Type
  (errors, types) = Map.mapEither bTranslationType gamma

-- /w ~> V_w
value :: Context -> Term -> Either TypeError Term
value context' m' = fix (normalize context' m')
where
  fix (Right n''') = valueHelper context' n'''
  fix (Left te) = Left te
  valueHelper :: Context -> Term -> Either TypeError Term
  valueHelper context (Pair (m, n)) = fix' tc1 tc2
  where
    tc1 :: Either TypeError Type
    tc1 = typeCheck context m
    tc2 :: Either TypeError Type
    tc2 = typeCheck context n
    fix' :: Either TypeError Type -> Either TypeError Type -> Either TypeError Term
    fix' (Right tau) (Right sigma)
      | tau == natb' && sigma == natb' = Right m
      | otherwise = Left (ValueError "occurred in value" (Pair (m, n)))
    fix' (Left te1) (Right _) = Left (ChainError1 "occurred in value (pair) / left" 21 te1)
    fix' (Right _) (Left te2) = Left (ChainError1 "occurred in value (pair) / right" 22 te2)
    fix' (Left te1) (Left te2) = Left (ChainError2 "occurred in value (pair) / both" 23 te1 te2)
  valueHelper context m = fix' tc
  where
    tc :: Either TypeError Type
    tc = typeCheck context m
    fix' :: Either TypeError Type -> Either TypeError Term
    fix' (Right tau)
      | tau == natb = Right (Value m)
      | otherwise = Left (ValueError "occurred in value" m)
    fix' (Left te1) = Left (ChainError1 "occurred in value" 24 te1)

```

```

-- /w ~> M_w
modulus :: Context -> Term -> Either TypeError Term
modulus context' m' = fix (normalize context' m')
where
  fix (Right n''') = modulusHelper context' n'''
  fix (Left te) = Left te
  modulusHelper :: Context -> Term -> Either TypeError Term
  modulusHelper context (Pair (m, n)) = fix' tc1 tc2
  where
    tc1 :: Either TypeError Type
    tc1 = typeCheck context m
    tc2 :: Either TypeError Type
    tc2 = typeCheck context n
    fix' :: Either TypeError Type -> Either TypeError Type -> Either TypeError Term
    fix' (Right tau) (Right sigma)
      | tau == natb' && sigma == natb' = Right n
      | otherwise = Left (ModulusError "occurred in modulus" (Pair (m, n)))
    fix' (Left te1) (Right _) = Left (ChainError1 "occurred in modulus (pair) / left" 25 te1)
    fix' (Right _) (Left te2) = Left (ChainError1 "occurred in modulus (pair) / right" 26 te2)
    fix' (Left te1) (Left te2) = Left (ChainError2 "occurred in modulus (pair) / both" 27 te1 te2)
  modulusHelper context m = fix' tc
  where
    tc :: Either TypeError Type
    tc = typeCheck context m
    fix' (Right tau)
      | tau == natb = Right (Modulus m)
      | otherwise = Left (ModulusError "occurred in modulus" m)
    fix' (Left te1) = Left (ChainError1 "occurred in modulus" 28 te1)

-- /Kleisli extension
ke :: Context -> Type -> Term -> Term -> Either TypeError Term
ke context''' tau'''' g f = keHelper (tc1, tc2) context''' tau'''' g f
where
  tc1 :: Either TypeError Type
  tc1 = typeCheck context''' g
  tc2 :: Either TypeError Type
  tc2 = typeCheck context''' f
  keHelper :: (Either TypeError Type, Either TypeError Type) -> Context -> Type -> Term
  keHelper -> Term -> Either TypeError Term
  keHelper (Right _, Left _) _ tau _ = Left (KleisliError "basic type check failed ~ 1" tau)
  keHelper (Left _, Right _) _ tau _ = Left (KleisliError "basic type check failed ~ 2" tau)
  keHelper (Left _, Left _) _ tau _ = Left (KleisliError "basic type check failed ~ 3" tau)
  keHelper (Right _, Right _) context' tau' g' f'
    | keTypeCheck context' tau' g' f' = keHelperHelper 0 context' tau' g' f'
    | otherwise = Left (KleisliError "complex type check failed" tau')
  where
    keTypeCheck :: Context -> Type -> Term -> Term -> Bool
    keTypeCheck context'''' rho' g' '''' f' '''' = fix2 tau sigma
    where

```

```

tau = typeCheck context "" g ""
sigma = typeCheck context "" f ""
fix1 :: Maybe (Type, Type) -> Type -> Either TypeError Type -> Bool
fix1 Nothing (Left _) = False
fix1 (Just (_, _)) (Left _) = False
fix1 Nothing (Right _) = False
fix1 (Just (tau1, tau2)) sigma' (Right rhob'')
  | tau1 == Nat && sigma' == natb && tau2 == rhob'' = True
  | otherwise = False
fix2 :: Either TypeError Type -> Either TypeError Type -> Bool
fix2 (Left _) (Right _) = False
fix2 (Right _) (Left _) = False
fix2 (Right tau'') (Right sigma') = fix1 (splitType tau'') sigma' (bTranslationType rho')
fix2 (Left _) (Left _) = False
keHelperHelper :: Integer -> Context -> Type -> Term -> Term -> Either TypeError Term
keHelperHelper version (Context gamma) (Function sigma tau) g f = ke'''
where
  fixK :: Either TypeError Type -> Either TypeError Term
  fixK (Right rho') = keHelperHelper (version + 1) (Context (Map.insert ("x" ++ show version) rho' gamma)) tau m f
  fixK (Left te) = Left (ChainError1 "occurred in ke, b-translation (type) of type sub-script (1)" 29 te)
  k :: Either TypeError Term
  k = fixK (bTranslationType sigma)
  fix :: Either TypeError Type -> Either TypeError Type -> Either TypeError Term
  fix (Right k') (Right rho') = Right (Abstract ("x" ++ show version, rho') k')
  fix (Left te) (Right _) = Left (ChainError1 "occurred in ke, ke recursion" 30 te)
  fix (Right _) (Left te) = Left (ChainError1 "occurred in ke, b-translation (type) of type sub-script (2)" 31 te)
  fix (Left te1) (Left te2) = Left (ChainError2 "occurred in ke, both" 32 te1 te2)
  ke''' :: Either TypeError Term
  ke''' = fix k (bTranslationType sigma)
  m :: Term
  m = Abstract ("k", Nat) (Apply (Apply g (Variable "k"))) (Variable ("x" ++ show version))
keHelperHelper _ (Product (tau, sigma)) _ _ = Left (KleisliError "invalid type" (Product (tau, sigma)))
keHelperHelper _ (Context gamma) Nat g f'''''' = fixPair v'' max'''
where
  context' = Context (Map.insert "&alpha;" (Function Nat Nat) gamma)
  mf :: Either TypeError Term
  mf = modulus context' f'''''' -- mf
  vf :: Either TypeError Term
  vf = value context' f'''''' -- vf
  fixVFA :: Either TypeError Term -> Either TypeError Term
  fixVFA (Right vf') = Right (Apply vf' (Variable "&alpha;"))
  fixVFA (Left te) = Left (ChainError1 "occurred in computing short" 33 te)
  vfa :: Either TypeError Term
  vfa = fixVFA vf -- vf alpha
  fixV'' :: Either TypeError Term -> Either TypeError Term
  fixV'' (Right vfa') = value context' (Apply g vfa')
  fixV'' (Left te) = Left (ChainError1 "occurred in computing v''" 34 te)
  v'' :: Either TypeError Term
  v'' = fixV'' vfa -- v(g(vf alpha))

```

```

fixM'' :: Either TypeError Term -> Either TypeError Term
fixM'' (Right vfa') = modulus context' (Apply g vfa')
fixM'' (Left te) = Left (ChainError1 "occurred in computing m''" 35 te)
m'' :: Either TypeError Term
m'' = fixM'' vfa -- m(g(vf alpha))
fixMax'' :: Either TypeError Term -> Either TypeError Term -> Either TypeError Term
fixMax'' (Right p) (Right q) = max' context' (Apply p (Variable "&alpha;")) (Apply q (Variable "&alpha;"))
fixMax'' (Left te) (Right _) = Left (ChainError1 "occured in computing b-translation (term), max / left" 36 te)
fixMax'' (Right _) (Left te) = Left (ChainError1 "occured in computing b-translation (term), max / right" 37 te)
fixMax'' (Left te1) (Left te2) = Left (ChainError2 "occured in computing b-translation (term), max / both" 38 te1 te2)
max'' :: Either TypeError Term
max'' = fixMax'' m'' mf -- max (m(g(vf &alpha;)) &alpha;, mf &alpha;)
fixPair :: Either TypeError Term -> Either TypeError Term -> Either TypeError Term
fixPair (Right p) (Right q) = Right (Pair (Abstract ("&alpha;", Function Nat Nat) (Apply p (Variable "&alpha;")), Abstract ("&alpha;", Function Nat Nat) q))
fixPair (Left te) (Right _) = Left (ChainError1 "occured in computing b-translation (term), pair / left" 39 te)
fixPair (Right _) (Left te) = Left (ChainError1 "occured in computing b-translation (term), pair / right" 40 te)
fixPair (Left te1) (Left te2) = Left (ChainError2 "occured in computing b-translation (term), pair / both" 41 te1 te2)

-- /t ~> t^b
bTranslationTerm :: Context -> Term -> Either TypeError Term
bTranslationTerm context' m' = bTranslationTermHelper (encodableTerm m') (typeCheck context' m') context' m'
where
  bTranslationTermHelper :: Bool -> Either TypeError Type -> Context -> Term -> Either TypeError Term
  bTranslationTermHelper False (Right _) _ m = Left (BTranslationTermError "unencodable term" m)
  bTranslationTermHelper True (Left te) _ _ = Left (ChainError1 "occurred in b-translation (early) (fails type check)" 0 te)
  bTranslationTermHelper False (Left te) _ _ = Left (ChainError1 "occurred in b-translation (early) (unencodable, fails type check)" 0 te)
  bTranslationTermHelper True (Right _) context m = bTranslationTermHelperHelper context m
  where
    bTranslationTermHelperHelper :: Context -> Term -> Either TypeError Term
    bTranslationTermHelperHelper (Variable x) = Right (Variable (bTranslationVar x))
    bTranslationTermHelperHelper (Context gamma) (Abstract (x, tau) m''') = fix x m''' tau'
    where
      m''' :: Either TypeError Term
      m''' = bTranslationTermHelperHelper (Context (Map.insert x tau gamma)) m'''
      tau' :: Either TypeError Type
      tau' = bTranslationType tau
      fix :: Var -> Either TypeError Term -> Either TypeError Type -> Either TypeError Term
      fix x' (Right m'') (Right tau'') = Right (Abstract (bTranslationVar x', tau'') m'')
      fix _ (Left te) (Right _) = Left (ChainError1 "occurred in b-translation (term), abstraction / body" 42 te)

```



```

fix _ (Right _) (Left te) = Left (ChainError1 "occurred in b-translation (term),
  abstraction / abstracted var type" 43 te)
fix _ (Left te1) (Left te2) = Left (ChainError2 "occurred in b-translation (term),
  abstraction / both" 44 te1 te2)
bTranslationTermHelperHelper context " (Apply m'''' n''') = fix m''' n'''
where
  m''' :: Either TypeError Term
  m''' = bTranslationTermHelperHelper context'' m''''
  n''' :: Either TypeError Term
  n''' = bTranslationTermHelperHelper context'' n''''
  fix :: Either TypeError Term -> Either TypeError Term -> Either TypeError Term
  fix (Right m'') (Right n'') = Right (Apply m'' n'')
  fix (Left te) (Right _) = Left (ChainError1 "occurred in b-translation (term),
    application / right" 45 te)
  fix (Right _) (Left te) = Left (ChainError1 "occurred in b-translation (term)
    application / left" 46 te)
  fix (Left te1) (Left te2) = Left (ChainError2 "occurred in b-translation (term)
    application / both" 47 te1 te2)
bTranslationTermHelperHelper _ Zero = Right (Pair (Abstract ("&alpha;", Function Nat
  Nat) Zero, Abstract ("&alpha;", Function Nat Nat) Zero))
bTranslationTermHelperHelper _ Succ = Right (Abstract ("x", natb) (Pair (Abstract
  ("&alpha;", Function Nat Nat) (Apply Succ (Apply val (Variable "&alpha;"))),
  Abstract ("&alpha;", Function Nat Nat) (Apply mod' (Variable "&alpha;")))))
where
  val = Value (Variable "x")
  mod' = Modulus (Variable "x")
bTranslationTermHelperHelper (Context _) (Rec rho) = fix tau ke'
where
  tau :: Either TypeError Type
  tau = bTranslationType rho
  fixContext :: Either TypeError Type -> Either TypeError Context
  fixContext (Right sigma) = Right (Context (Map.fromList [("a", sigma), ("&alpha;",
    Function Nat Nat), ("k", Nat), ("h", natb), ("f", Function natb (Function sigma
    sigma)))))
  fixContext (Left te) = Left (ChainError1 "occurred in computing the context" 48 te)
  context'' :: Either TypeError Context
  context'' = fixContext tau
  fix' :: Either TypeError Type -> Either TypeError Term
  fix' (Right sigma) = Right (Apply (Apply (Rec sigma) (Variable "a"))) (Abstract ("k",
    Nat) (Apply (Variable "f") (Pair (p1, p2)))))
  fix' (Left te) = Left (ChainError1 "occurred in computing first argument for ke
    extension" 49 te)
  g :: Either TypeError Term
  g = fix' tau
  f :: Term
  f = Variable "h"
  p1 :: Term
  p1 = Abstract ("&alpha;", Function Nat Nat) (Variable "k")
  p2 :: Term
  p2 = Abstract ("&alpha;", Function Nat Nat) Zero
  fixKe :: Either TypeError Term -> Either TypeError Context -> Either TypeError
    Term
  fixKe (Right g') (Right context'') = ke context'' rho g' f
  fixKe (Left te) (Right _) = Left (ChainError1 "occurred in computing ke /
    context" 50 te)

```

```

fixKe (Right _) (Left te) = Left (ChainError1 "occurred in computing ke / rho^b"
  51 te)
fixKe (Left te1) (Left te2) = Left (ChainError2 "occurred in computing ke / both"
  52 te1 te2)
ke' :: Either TypeError Term
ke' = fixKe g context''
fix :: Either TypeError Type -> Either TypeError Term -> Either TypeError Term
fix (Right sigma) (Right ke'') = Right (Abstract ("a", sigma) (Abstract ("f",
  Function natb (Function sigma sigma)) (Abstract ("h", natb) ke'')))
fix (Right _) (Left te) = Left (ChainError1 "occurred in b-translation (term), rec
  / ke" 53 te)
fix (Left te) (Right _) = Left (ChainError1 "occurred in b-translation (term), rec
  / rho^b" 54 te)
fix (Left te1) (Left te2) = Left (ChainError2 "occurred in b-translation (term),
  rec" 55 te1 te2)
bTranslationTermHelperHelper context'' m'' = fix tc
where
  tc :: Either TypeError Type
  tc = typeCheck context'' m''
  fix :: Either TypeError Type -> Either TypeError Term
  fix (Left te) = Left (ChainError1 "occurred in b-translation (term), unencodable
    term" 56 te)
  fix (Right _) = Left (BTranslationTermError "unencodable term for b-translation"
    m)

```

```
bTranslationTermNormal :: Context -> Term -> Either TypeError Term
```

```
bTranslationTermNormal context term = fix context' original
```

```

where
  context' :: Either TypeError Context
  context' = bTranslationContext context
  original :: Either TypeError Term
  original = bTranslationTerm context term
  fix :: Either TypeError Context -> Either TypeError Term -> Either TypeError Term
  fix (Left te1) (Left te2) = Left (ChainError2 "occurred in b-translation (term, normal) /
    both" 1001 te1 te2)
  fix (Right _) (Left te) = Left (ChainError1 "occurred in b-translation (term, normal) /
    context" 1002 te)
  fix (Left te) (Right _) = Left (ChainError1 "occurred in b-translation (term, normal) /
    term" 1003 te)
  fix (Right context'') (Right m) = normalize context'' m

```

```
bTranslationTermOmega :: Context -> Term -> Either TypeError Term
```

```
bTranslationTermOmega context term = bTranslationTermOmegaHelper (typeCheck context term)
  context term
```

```

where
  bTranslationTermOmegaHelper :: Either TypeError Type -> Context -> Term -> Either
    TypeError Term
  bTranslationTermOmegaHelper (Left te) context term = Left te
  bTranslationTermOmegaHelper (Right tau) context term
  | tau == Function (Function Nat Nat) Nat = bTranslationTermOmegaHelperHelper context term
  | otherwise = Left (ModulusOfContinuityError "occurred in modulus of continuity" tau)
where
  bTranslationTermOmegaHelperHelper :: Context -> Term -> Either TypeError Term
  bTranslationTermOmegaHelperHelper context term = fix2 context' normal
  where

```

```

omega :: Term
omega = Abstract ("f", natb) (Pair (pLeft, pRight))
pLeft :: Term
pLeft = Abstract ("&alpha;", Function Nat Nat) (Apply (Variable "&alpha;") (Apply
  (Value (Variable "f")) (Variable "&alpha;")))
mLeft :: Term
mLeft = Apply (Modulus (Variable "f")) (Variable "&alpha;")
mRight :: Term
mRight = Apply Succ (Apply (Value (Variable "f")) (Variable "&alpha;"))
pRight :: Term
pRight = Abstract ("&alpha;", Function Nat Nat) (Max mLeft mRight)
context' :: Either TypeError Context
context' = bTranslationContext context
original :: Either TypeError Term
original = bTranslationTerm context term
fix1 :: Either TypeError Context -> Either TypeError Term -> Either TypeError
  Term
fix1 (Left te1) (Left te2) = Left (ChainError2 "occurred in modulus of continuity /
  both" 2001 te1 te2)
fix1 (Right _) (Left te) = Left (ChainError1 "occurred in modulus of continuity /
  context" 2002 te)
fix1 (Left te) (Right _) = Left (ChainError1 "occurred in modulus of continuity /
  term" 2003 te)
fix1 (Right context') (Right m) = normalize context' (Apply m omega)
normal :: Either TypeError Term
normal = fix1 context' original
fix2 :: Either TypeError Context -> Either TypeError Term -> Either TypeError
  Term
fix2 (Left te1) (Left te2) = Left (ChainError2 "occurred in modulus of continuity /
  both" 2001 te1 te2)
fix2 (Right _) (Left te) = Left (ChainError1 "occurred in modulus of continuity /
  context" 2002 te)
fix2 (Left te) (Right _) = Left (ChainError1 "occurred in modulus of continuity /
  term" 2003 te)
fix2 (Right context') (Right m) = Right (reassign context' (reassign context " m))

modulusOfContinuity :: Context -> Term -> Either TypeError Term
modulusOfContinuity context term = modulusOfContinuityHelper (typeCheck context term) context
  term
where
  modulusOfContinuityHelper :: Either TypeError Type -> Context -> Term -> Either
    TypeError Term
  modulusOfContinuityHelper (Left te) context term = Left te
  modulusOfContinuityHelper (Right tau) context term
    | tau == Function (Function Nat Nat) Nat = modulusOfContinuityHelperHelper context term
    | otherwise = Left (ModulusOfContinuityError "occurred in modulus of continuity" tau)
  where
    modulusOfContinuityHelperHelper :: Context -> Term -> Either TypeError Term
    modulusOfContinuityHelperHelper context term = fix2 context' normal
    where
      omega :: Term
      omega = Abstract ("f", natb) (Pair (pLeft, pRight))
      pLeft :: Term
      pLeft = Abstract ("&alpha;", Function Nat Nat) (Apply (Variable "&alpha;") (Apply
        (Value (Variable "f")) (Variable "&alpha;")))
      mLeft :: Term

```

```

mLeft = Apply (Modulus (Variable "f")) (Variable "&alpha;")
mRight :: Term
mRight = Apply Succ (Apply (Value (Variable "f")) (Variable "&alpha;"))
pRight :: Term
pRight = Abstract ("&alpha;", Function Nat Nat) (Max mLeft mRight)
context' :: Either TypeError Context
context' = bTranslationContext context
original :: Either TypeError Term
original = bTranslationTerm context term
fix1 :: Either TypeError Context -> Either TypeError Term -> Either TypeError
  Term
fix1 (Left te1) (Left te2) = Left (ChainError2 "occurred in modulus of continuity /
  both" 2001 te1 te2)
fix1 (Right _) (Left te) = Left (ChainError1 "occurred in modulus of continuity /
  context" 2002 te)
fix1 (Left te) (Right _) = Left (ChainError1 "occurred in modulus of continuity /
  term" 2003 te)
fix1 (Right context') (Right m) = normalize context' (Modulus (Apply m omega))
normal :: Either TypeError Term
normal = fix1 context' original
fix2 :: Either TypeError Context -> Either TypeError Term -> Either TypeError
  Term
fix2 (Left te1) (Left te2) = Left (ChainError2 "occurred in modulus of continuity /
  both" 2001 te1 te2)
fix2 (Right _) (Left te) = Left (ChainError1 "occurred in modulus of continuity /
  context" 2002 te)
fix2 (Left te) (Right _) = Left (ChainError1 "occurred in modulus of continuity /
  term" 2003 te)
fix2 (Right context') (Right m) = Right (reassign context' m)

```

---

EXAMPLES

---

— */Empty context (used for closed terms)*

```

empty :: Context
empty = Context Map.empty

```

— */Addition example*

```

term00 :: Term
term00 = Abstract ("x", Nat) $ Abstract ("y", Nat) $ Apply (Apply (Apply (Rec Nat) $ Variable
  "x") sub) $ Variable "y"
  where
    sub :: Term
    sub = Abstract ("u", Nat) $ Abstract ("v", Nat) $ Apply Succ $ Variable "v"
context00 :: Context
context00 = empty
r :: Term
r = term00

```

— */Example 01*

```

term01 :: Term
term01 = Apply (Abstract ("x", Function Nat Nat) $ Variable "x") $ Variable "z"
context01 :: Context
context01 = Context (Map.fromList [("z", Function Nat Nat)])

```

```

-- /Example 02
term02 :: Term
term02 = Apply (Abstract ("x", Function Nat Nat) (Variable "x")) (Abstract ("x", Nat) (Variable
"x"))
context02 :: Context
context02 = empty

-- /Example 03 (numeral 3)
term03 :: Term
term03 = numeral 3
context03 :: Context
context03 = empty

-- /Example 04
term04 :: Term
term04 = Rec Nat
context04 :: Context
context04 = empty

-- /Example 05
term05 :: Term
term05 = Apply (Abstract ("x", Nat) (Pair (Variable "x", Variable "x"))) (Variable "z")
context05 :: Context
context05 = empty

-- /Example 06
term06 :: Term
term06 = Apply (Apply (Apply (Rec Nat) (Apply Succ Zero)) (Variable "h")) Zero
context06 :: Context
context06 = Context (Map.fromList [("h", Function Nat (Function Nat Nat))])

-- /Example 07 (reduces to numeral (c + d)),
c :: Integer
d :: Integer
(c, d) = (3, 10) -- as it appears in the slides
term07 :: Term
term07 = Apply (Apply r (numeral c)) (numeral d)
context07 :: Context
context07 = empty

-- /Example 08
term08 :: Term
term08 = Max t s
where

```

```

t = Apply (Abstract ("x", Nat) (Variable "x")) (Variable "b")
s = Variable "b"
context08 :: Context
context08 = Context (Map.fromList [("b", Nat)])

-- /Example 09
term09 :: Term
term09 = Apply (Variable "b") (Apply (Variable "b") (Apply (Variable "b") (Apply (Variable "a")
(Variable "b"))))
context09 :: Context
context09 = Context (Map.fromList [("b", Function Nat Nat), ("a", Nat)])

-- /Example 10
term10 :: Term
term10 = Rec (Function Nat (Function Nat Nat))
context10 :: Context
context10 = Context (Map.fromList [("b", Function Nat Nat), ("a", Nat)])

-- /Example 11
term11 :: Term
term11 = Abstract ("h", Nat) (Apply (Apply r (Variable "h")) (numeral 5))
context11 :: Context
context11 = empty

-- /Addition example
term12 :: Term
term12 = Abstract ("x", Nat) $ Abstract ("y", Nat) $ Apply (Apply (Apply (Rec Nat) $ Variable
"x") sub) $ Variable "y"
where
  sub :: Term
  sub = Abstract ("u", Nat) $ Abstract ("v", Nat) $ Apply (Apply r $ Variable "v") $ Variable
"u"
context12 :: Context
context12 = empty

term13 :: Term
term13 = Apply (Apply term12 $ numeral c) $ numeral d
context13 :: Context
context13 = empty

term14 :: Term
term14 = Abstract ("a", Function Nat Nat) Zero
context14 :: Context
context14 = empty

```

## A.2 File: server.js

```
"use strict";
var http = require('http');
const execSync = require('child_process').execSync;

function runShellCommand(arg) {
  console.log(arg);
  var r = execSync(core.exe $arg.replace(/(?:\r\n|\\r\\n)/g, " "));
  var s = r.toString();
  var x = s.toString().replace(/(?:\r\n|\\r\\n)/g, " ").replace(/(?:\\)/g, '\\');
  return x.slice(1, x.length - 1);
}

http.createServer(function(req, res) {
  res.setHeader('Content-Type', 'text/json');
  res.setHeader('Access-Control-Allow-Origin', '*');
```

```
    res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, PATCH, DELETE'); // If needed
    res.setHeader('Access-Control-Allow-Headers', 'X-Requested-With,content-type'); // If needed
    res.setHeader('Access-Control-Allow-Credentials', true);
    res.writeHead(200);
    req.on('data', function(message) {
      var m = message.toString().replace(/(?:\r\n|\\r\\n)/g, " ");
      console.log("m", m);
      var r = runShellCommand($m);
      res.write($r);
    });
    req.on('end', function() {
      res.end();
    });
  }).listen(8000);
```

## A.3 File: blocks.js

```

Blockly.Blocks['set_term'] = {
  init: function () {
    this.appendValueInput("NAME")
      .setCheck("Term");
    this.appendField("term: ");
    this.setInputsInline ( false );
    this.setColour(400);
    this.setDeletable ( false );
    this.setTooltip ( "Term parent" );
    this.setHelpUrl ( "" );
  }
};

Blockly.Blocks['zero'] = {
  init: function () {
    this.appendDummyInput()
      .appendField("0");
    this.setInputsInline ( true );
    this.setOutput(true, "Term");
    this.setColour(180);
    this.setTooltip ( "Zero" );
    this.setHelpUrl ( "" );
  }
};

Blockly.Blocks['variable'] = {
  init: function () {
    this.appendDummyInput()
      .appendField(new Blockly.FieldTextInput (""), "VAR");
    this.setInputsInline ( true );
    this.setOutput(true, "Term");
    this.setColour(300);
    this.setTooltip ( "Variable" );
    this.setHelpUrl ( "" );
  }
};

Blockly.Blocks['abstraction'] = {
  init: function () {
    this.appendDummyInput()
      .appendField("λ")
      .appendField("(")
      .appendField(new Blockly.FieldTextInput (""), "ABVAR")
      .appendField(":");
    this.appendValueInput("ABTYPE")
      .setCheck("Type");
    this.appendDummyInput()
      .appendField(")")
      .appendField(".");
    this.appendValueInput("BODY")
      .setCheck("Term");
    this.appendDummyInput();
    this.setInputsInline ( true );
    this.setOutput(true, "Term");
    this.setColour(340);
  }
};

```

```

    this.setTooltip ( "Abstraction" );
    this.setHelpUrl ( "" );
  }
};

Blockly.Blocks['application'] = {
  init: function () {
    this.appendValueInput("LEFT")
      .setCheck("Term");
    this.appendValueInput("RIGHT")
      .setCheck("Term");
    this.setInputsInline ( true );
    this.setOutput(true, "Term");
    this.setColour(200);
    this.setTooltip ( "Application" );
    this.setHelpUrl ( "" );
  }
};

Blockly.Blocks['recursor'] = {
  init: function () {
    this.appendDummyInput()
      .appendField("rec");
    this.appendValueInput("RECTYPE")
      .setCheck("Type");
    this.setInputsInline ( true );
    this.setOutput(true, "Term");
    this.setColour(260);
    this.setTooltip ( "Recursor" );
    this.setHelpUrl ( "" );
  }
};

Blockly.Blocks['successor'] = {
  init: function () {
    this.appendDummyInput()
      .appendField("succ");
    this.setInputsInline ( true );
    this.setOutput(true, "Term");
    this.setColour(220);
    this.setTooltip ( "Successor" );
    this.setHelpUrl ( "" );
  }
};

Blockly.Blocks['nat'] = {
  init: function () {
    this.appendDummyInput()
      .appendField("ℕ");
    this.setInputsInline ( false );
    this.setOutput(true, "Type");
    this.setColour(20);
    this.setTooltip ( "Naturals" );
    this.setHelpUrl ( "" );
  }
};

```

```

Blockly.Blocks['function'] = {
  init: function() {
    this.appendValueInput("DOM")
      .setCheck("Type");
    this.appendDummyInput()
      .appendField(new Blockly.FieldLabelSerializable("→"), "ARROW");
    this.appendValueInput("COD")
      .setCheck("Type");
    this.setInputsInline(true);
    this.setOutput(true, "Type");
    this.setColour(60);
    this.setTooltip("Arrow");
    this.setHelpUrl("");
  }
};

Blockly.Blocks['set_context'] = {
  init: function() {
    this.declarationCount_ = 3;
    this.updateShape_();
    this.setDeletable(false);
    this.setMutator(new Blockly.Mutator(['set_context_declaration']));
    this.setColour(450);
    this.setTooltip('Context');
  },
  mutationToDom: function() {
    var container = Blockly.utils.xml.createElement('mutation');
    container.setAttribute('declarations', this.declarationCount_);
    return container;
  },
  domToMutation: function(container) {
    this.declarationCount_ = parseInt(container.getAttribute('declarations'), 10);
    this.updateShape_();
  },
  decompose: function(workspace) {
    var containerBlock = workspace.newBlock('set_context_container');
    containerBlock.initSvg();
    var connection = containerBlock.getInput('STACK').connection;
    for (var i = 0; i < this.declarationCount_; i++) {
      var declarationBlock = workspace.newBlock('set_context_declaration');
      declarationBlock.initSvg();
      connection.connect(declarationBlock.previousConnection);
      connection = declarationBlock.nextConnection;
    }
    return containerBlock;
  },
  compose: function(containerBlock) {
    var declarationBlock = containerBlock.getInputTargetBlock('STACK');
    // Count number of inputs.
    var connections = [];
    var data = []
    while (declarationBlock && !declarationBlock.isInsertionMarker()) {
      connections.push(declarationBlock.valueConnection_);
      data.push(declarationBlock.userData_);
      declarationBlock = declarationBlock.nextConnection &&
        declarationBlock.nextConnection.targetBlock();
    }
  },

```

```

    }
    // Disconnect any children that don't belong.
    for (var i = 0; i < this.itemCount_; i++) {
      var connection = this.getInput('ADD' + i).connection.targetConnection;
      if (connection && connections.indexOf(connection) == -1) {
        connection.disconnect();
      }
    }
    for (var i = 0; i < this.optionCount_; i++) {
      this.setFieldValue(data[i] || 'option', 'USER' + i);
    }
    this.declarationCount_ = connections.length;
    this.updateShape_();
    // Reconnect any child blocks.
    for (var i = 0; i < this.itemCount_; i++) {
      Blockly.Mutator.reconnect(connections[i], this, 'ADD' + i);
    }
  },
  saveConnections: function(containerBlock) {
    // Store names and values for each option.
    var declarationBlock = containerBlock.getInputTargetBlock('STACK');
    var i = 0;
    while (declarationBlock) {
      var input = this.getInput('ADD' + i);
      declarationBlock.userData_ = this.getFieldValue('USER' + i);
      declarationBlock.valueConnection_ = input && input.connection.targetConnection;
      i++;
      declarationBlock = declarationBlock.nextConnection &&
        declarationBlock.nextConnection.targetBlock();
    }
  },
  updateShape_: function() {
    // if (this.declarationCount_ && this.getInput('EMPTY')) {
    //   this.removeInput('EMPTY');
    // } else if (!this.itemCount_ && !this.getInput('EMPTY')) {
    //   this.appendDummyInput('EMPTY')
    //   .appendField(Blockly.Msg['LISTS_CREATE_EMPTY_TITLE']);
    // }
    // Add new inputs.
    for (var i = 0; i < this.declarationCount_; i++) {
      if (!this.getInput('ADD' + i)) {
        var input = this.appendValueInput('ADD' + i).setCheck('Type')
          .appendField(new Blockly.FieldTextInput(""), 'USER' + i)
          .appendField(':', '');
        .setAlign(Blockly.ALIGN_RIGHT);
        if (i == 0) {
          input.appendField("");
        }
      }
    }
    // Remove deleted inputs.
    while (this.getInput('ADD' + i)) {
      this.removeInput('ADD' + i);
      i++;
    }
  },

```

```

    onchange: function () {
    }
}

Blockly.Blocks['set_context_container'] = {
  /**
   * Mutator block for list container.
   * @this {Blockly.Block}
   */
  init: function () {
    this.appendDummyInput()
      .appendField('context');
    this.appendStatementInput('STACK');
    this.setColour(450);
    this.setTooltip('Add declarations here');
    this.contextMenu = false;
  }
};

```

```

    }
  };

  Blockly.Blocks['set_context_declaration'] = {
    /**
     * Mutator block for adding items.
     * @this {Blockly.Block}
     */
    init: function () {
      this.appendDummyInput()
        .appendField('declaration');
      this.setPreviousStatement(true);
      this.setNextStatement(true);
      this.setColour(450);
      this.setTooltip('Add/Remove this to the context');
      this.contextMenu = false;
    }
  };
};

```

## A.4 File: index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>CM30082 | Modulus of Continuity</title>
    <script src="https://unpkg.com/blockly/blockly.min.js"></script>
    <script src="blocks.js"></script>
    <style>
      #display * {
        vertical-align: middle;
      }
      #display {
        font-family: 'Courier New', monospace;
        padding: 10px;
        margin: 20px;
        width: 1200px;
        height: 250px;
        border: thin solid black;
      }
      body {
        margin: 0 10%;
        background-color: #fff;
        font-family: sans-serif;
      }
      h1 {
        font-weight: normal;
        font-size: 140%;
      }
      td {
        padding: 1ex;
      }
      img {
        border: none;
      }
    </style>
  </head>
  <body onload="start()">
    <h1>CM30082: Individual project</h1>
    <h2>Inductive functions in System T: Modulus of continuity</h2>
    
    
    <p>
      This is my CM30082 project about performing operations on lambda-terms in System
      <em>T</em>.
    </p>
    <table>
      <tr>
        <td>
          <div id="blocklyDiv" style="width: 1200px; height: 400px;"></div>
        </td>
      </tr>
    </table>
  </body>
</html>
```

```
</tr>
<tr>
  <td>
    <input type="button" value="Echo" onclick="echoCall()" id="echoButton">
    <input type="button" value="Normal form" onclick="normalizeCall()"
      id="normalizeButton">
    <input type="button" value="Modulus of continuity" onclick="modulusCall()"
      id="modulusButton">
    <input type="button" value="Type check" onclick="typeCheckCall()"
      id="typeCheckButton">
    <!-- <input type="button" value="b-translation" onclick="bTranslationCall()"
      id="bTranslationButton">
    <input type="button" value="b-translation (omega)"
      onclick="bTranslationOmegaCall()" id="bTranslationOmegaButton"> -->
  </td>
</tr>
<tr>
  <td>
    <div id="display"
      style="width: 1000px; height: 200px;"></div>
  </td>
</tr>
<tr>
  <td>
    <input type="button" id="export" value="Export to XML" onclick="toXml()">
    <input type="button" id="import" value="Import from XML"
      onclick="fromXml()">
  </td>
</tr>
<tr>
  <td>
    <textarea id="importExport"
      style="width: 600px; height: 200px;" onchange="textAreaChange();"
      onkeyup="textAreaChange()"></textarea>
  </td>
</tr>
</table>
<xml xmlns="https://developers.google.com/blockly/xml" id="toolbox" style="display:
  none">
  <category name="Types" colour="200">
    <block type="nat"></block>
    <block type="function"></block>
  </category>
  <category name="Terms" colour="160">
    <block type="variable"></block>
    <block type="abstraction"></block>
    <block type="application"></block>
    <block type="zero"></block>
    <block type="successor"></block>
    <block type="recursor"></block>
  </category>
</xml>
</script>
```



```

function ClearEditor(form) {
  $scope.scriptName = '';
  $scope.scriptDescription = '';
  $scope.userDomXml = '';
  $scope.userJavascriptCode = '';
  //var xml = Blockly.Xml.textToDom('<xml
    xmlns="http://www.w3.org/1999/xhtml"></xml>');
  //Blockly.Xml.domToWorkspace(Blockly.mainWorkspace, xml);
  Blockly.mainWorkspace.clear();
  // discard ();
}

function toXml() {
  var output = document.getElementById('importExport');
  var xml = Blockly.Xml.workspaceToDom(workspace);
  output.value = Blockly.Xml.domToPrettyText(xml);
  output.focus();
  output.select();
}

function fromXml() {
  workspace.clear();
  var input = document.getElementById('importExport');
  var xml = Blockly.Xml.textToDom(input.value);
  Blockly.Xml.domToWorkspace(xml, workspace);
}

function testContext(str) {
  try {
    var parsed = JSON.parse(str);
    var decs = parsed['context']['declarations'];
    var vn = unique(decs.map(a => a['vn']));
    var t = decs.map(a => a['t']);
    if (vn.length == t.length) {
      return true;
    } else {
      return false;
    }
  } catch (Exception) {
    return false;
  }
}

return true;
}

function testJSON(str) {
  try {
    JSON.parse(str);
  } catch (e) {
    return false;
  }
}

return true;
}

function appendDom() {
  var blocks = document.getElementById('workspace-blocks');

```

```

    if (blocks.firstElementChild) {
      Blockly.Xml.appendDomToWorkspace(blocks, workspace);
    }
  }

  const generator = new Blockly.Generator('JSON');
  generator.PRECEDENCE = 0;
  generator['set_context'] = function(block) {
    var arr = [... Array(block.childBlocks_.length).keys()];
    var var_names = block.inputList.map(x => x.fieldRow[0].value_);
    var types = arr.map(b => generator.valueToCode(block, 'ADD' + b,
      generator.PRECEDENCE));
    context = '\context\':{\declarations\':[' + arr.map(b => '{\vn\':" + var_names[b]
      + '\",\t\':" + types[b] + '\'}).join(',') + '],\'
    console.log(context);
    return context;
  };
  generator['set_term'] = function(block) {
    var value_name = generator.valueToCode(block, 'NAME', generator.PRECEDENCE);
    var code = '\term\':" + value_name + \'';
    return code;
  };
  generator['nat'] = function(block) {
    return ['{\category\':"NAT\"'}, generator.PRECEDENCE];
  };
  generator['function'] = function(block) {
    var value_dom = generator.valueToCode(block, 'DOM', generator.PRECEDENCE);
    var value_cod = generator.valueToCode(block, 'COD', generator.PRECEDENCE);
    var code = '{\category\':"FUNCTION\",\'dom\':" + value_dom + ',\'cod\':" + value_cod
      + \'';
    return [code, generator.PRECEDENCE];
  };
  generator['variable'] = function(block) {
    var text_var = block.getFieldValue('VAR');
    var code = '{\category\':"VARIABLE\",\'var\':" + text_var + \'';
    return [code, generator.PRECEDENCE];
  };
  generator['abstraction'] = function(block) {
    var text_abvar = block.getFieldValue('ABVAR');
    var value_abtype = generator.valueToCode(block, 'ABTYPE', generator.PRECEDENCE);
    var value_body = generator.valueToCode(block, 'BODY', generator.PRECEDENCE);
    var code = '{\category\':"ABSTRACT\",\'abvar\':" + text_abvar + '\",\'abtype\':" +
      value_abtype + '\",\'body\':" + value_body + \'';
    return [code, generator.PRECEDENCE];
  };
  generator['application'] = function(block) {
    var value_left = generator.valueToCode(block, 'LEFT', generator.PRECEDENCE);
    var value_right = generator.valueToCode(block, 'RIGHT', generator.PRECEDENCE);
    var code = '{\category\':"APPLY\",\'left\':" + value_left + '\",\'right\':" + value_right +
      \'';
    return [code, generator.PRECEDENCE];
  };
  generator['recursor'] = function(block) {
    var value_rectype = generator.valueToCode(block, 'RECTYPE', generator.PRECEDENCE);
    var code = '{\category\':"REC\",\'rectype\':" + value_rectype + \'';
    return [code, generator.PRECEDENCE];
  };

```

```

};
generator['successor'] = function(block) {
  var code = '{\\"category\\":\\"SUCC\\"}';
  return [code, generator.PRECEDENCE];
};
generator['zero'] = function(block) {
  var code = '{\\"category\\":\\"ZERO\\"}';
  return [code, generator.PRECEDENCE];
};
generator.scrub_ = function(block, code, opt_thisOnly) {
  const nextBlock =
    block.nextConnection && block.nextConnection.targetBlock();
  const nextCode =
    opt_thisOnly ? '' : generator.blockToCode(nextBlock);
  return code + nextCode;
};

function size(json) {
  switch (json["category"]) {
    case "NAT":
      return 1;
    case "FUNCTION":
      return 2;
    case "PRODUCT":
      return 2;
    case "VARIABLE":
      return 1;
    case "SUCC":
      return 1;
    case "ZERO":
      return 1;
    case "APPLY":
      return 2;
    case "ABSTRACT":
      return 2;
    case "VALUE":
      return 1;
    case "MODULUS":
      return 1;
    case "MAX":
      return 1;
    case "PAIR":
      return 1;
  }
  return -1;
}

function isNumeric(str){
  return /\^d+$/ .test(str);
}

function isLetter(str) {
  return str.length === 1 && str.match(/[a-z]/i);
}

function subscriptHandlerNumeric(s) {
  var ar = s.split('');
  var nums = ar.filter(isNumeric);

```

```

  var letters = ar.filter(isLetter);
  console.log(nums);
  console.log(typeof(nums));
  if (nums.length == 0) {
    return '<mi>' + s + '</mi>';
  } else if (s === (letters.join('') + nums.join(''))) {
    return '<msub><mi>' + letters.join('') + '</mi><mn>' + nums.join('') +
      '</mn></msub>';
  } else {
    return '<mi>' + s + '</mi>';
  }
}

function superscriptHandler(s) {
  if (s.endsWith('^b')) {
    var t = s.replace('^b', '');
    return '<msup>' + subscriptHandlerNumeric(t) + '<mi>' +
      mathvariant="normal">b</mi></msup>';
  } else {
    return subscriptHandlerNumeric(s);
  }
}

function toHTML(json){
  console.log('category', json.hasOwnProperty('category'), json);
  switch(json["category"]) {
    case "OKAY":
      return '<math>' + toHTML(json['ok']) + '</math>';
    case "ERROR":
      return '<ul>' + toHTML(json['err']) + '</ul>';
    case "CONTAINERERROR":
      return '<li>container error</li>';
    case "VARNOTFOUNDERERROR":
      return '<li>var not found error: <math>' + superscriptHandler(json['var']) +
        '</math></li>';
    case "APPLYERROR":
      return '<li>application error: <math>' + toHTML(json['m']) + ':\&nbsp;' +
        toHTML(json['tau']) + '</math>, &nbsp;<math>' + toHTML(json['n']) + ':\&nbsp;' +
        toHTML(json['sigma']) + '</math> </li>';
    case "MAXERROR":
      return '<li>max error: <math>' + toHTML(json['m']) + '</math>, <math>' +
        toHTML(json['n']) + '</math></li>';
    case "VALUEERROR":
      return '<li>value error: <math>' + toHTML(json['m']) + '</math></li>';
    case "MODULUSERERROR":
      return '<li>modulus error: <math>' + toHTML(json['m']) + '</math></li>';
    case "BTRANSLATIONTYPEERROR":
      return '<li>b-translation (type) error: <math>' + toHTML(json['m']) +
        '</math></li>';
    case "BTRANSLATIONTERMERROR":
      return '<li>b-translation (term) error: <math>' + toHTML(json['m']) + '</math>,
        <math>' + toHTML(json['n']) + '</math></li>';
    case "KLEISLIERROR":
      return '<li>kleisli error: <math>' + toHTML(json['tau']) + '</math></li>';
    case "MODULUSOFCONTINUITYERROR":
      return '<li>modulus of continuity error: <math>' + toHTML(json['tau']) +
        '</math></li>';

```

```

case "CHAINERROR1":
  return '<li>' + json['message'] + '<ul>' + toHTML(json['te']) + '</ul></li>'
case "CHAINERROR2":
  return '<li>' + json['message'] + '<ul>' + toHTML(json['te1']) + toHTML(json['te2']) +
    '</ul></li>'
case "CHAINERRORMANY":
  return '<li>' + json['message'] + '<ul>' + json['errors'].map(toHTML).join('') +
    '</ul></li>'
case "NAT":
  return '<mi mathvariant="double-struck">N</mi>';
case "NATB":
  return '<msup><mi mathvariant="double-struck">N</mi><mi
    mathvariant="normal">b</mi></msup>';
case "FUNCTION":
  var s;
  if (size(json['dom']) > 1) {
    s = '<mo></mo>' + toHTML(json['dom']) + '<mo></mo>'
  } else {
    s = toHTML(json['dom'])
  }
  var t;
  if (size(json['cod']) > 1) {
    t = '<mo></mo>' + toHTML(json['cod']) + '<mo></mo>'
  } else {
    t = toHTML(json['cod'])
  }
  return s + '&nbsp;<mo>&rarr;</mo>&nbsp;' + t;
case "PRODUCT":
  var s;
  if (size(json['left']) > 1) {
    s = '<mo></mo>' + toHTML(json['left']) + '<mo></mo>'
  } else {
    s = toHTML(json['left'])
  }
  var t;
  if (size(json['right']) > 1) {
    t = '<mo></mo>' + toHTML(json['right']) + '<mo></mo>'
  } else {
    t = toHTML(json['right'])
  }
  return s + '&nbsp;<mo>&times;</mo>&nbsp;' + t;
case "VARIABLE":
  return superscriptHandler(json['var']);
case "SUCC":
  return '<mi mathvariant="normal">succ</mi>';
case "ZERO":
  return '<mn>0</mn>';
case "REC":
  return '<msub><mi mathvariant="normal">rec</mi><mn>' + toHTML(json['rectype'])
    + '</mn></msub>';
case "APPLY":
  var s;
  if (size(json['left']) > 1) {
    s = '<mo></mo>' + toHTML(json['left']) + '<mo></mo>'
  } else {
    s = toHTML(json['left'])

```

```

  }
  var t;
  if (size(json['right']) > 1) {
    t = '<mo></mo>' + toHTML(json['right']) + '<mo></mo>'
  } else {
    t = toHTML(json['right'])
  }
  return s + '&nbsp;' + t;
case "ABSTRACT":
  return '<mi>&lambda;</mi>(' + superscriptHandler(json['abvar']) + '&nbsp;' +
    toHTML(json['abtype']) + '&nbsp;<mo></mo>&nbsp;' + toHTML(json['body']));
case "MAX":
  return '<mi mathvariant="normal">max</mi>(' + toHTML(json['left']) + '&nbsp;' +
    toHTML(json['right']) + ')';
case "VALUE":
  return '<msub><mi mathvariant="normal">V</mi><mn>' + toHTML(json['content'])
    + '</mn></msub>';
case "MODULUS":
  return '<msub><mi mathvariant="normal">M</mi><mn>' + toHTML(json['content'])
    + '</mn></msub>';
case "PAIR":
  return '<mo>&lang;</mo>' + toHTML(json['left']) + '<mo></mo>&nbsp;' +
    toHTML(json['right']) + '<mo>&rang;</mo>';
}
}
workspace = [];
function start() {
  workspace = Blockly.inject('blocklyDiv', options);
  appendDom();
  workspace.addChangeListener(Blockly.Events.disableOrphans);
  this.workspace.scrollCenter();
}
function echoCall () {
  compute('E');
}
function normalizeCall () {
  compute('N');
}
function modulusCall () {
  compute('M');
}
function bTranslationCall () {
  compute('B');
}
function bTranslationOmegaCall () {
  compute('O');
}
function typeCheckCall () {
  compute('T');
}
function compute(command) {
  var output = document.getElementById('display');
  var json = generator.workspaceToCode(workspace).toString();
  const xhr = new XMLHttpRequest();
  pre = ('{"category":"INPUTSTRUCTURE","command":"' + command + '\",' +
    json.slice(0, json.length - 1) + ''}).replace(/(?:\r\n|\r|\n)/g, '');

```

```

console.log('pre ', pre);
if (!testJSON(pre)) {
  console.log('error1: ', pre);
  mathHandler("<ul><li>construction error:<ul><li>fill all the gaps in the  
term/context</li></ul></li></ul>");
  output.focus();
  return;
}
try {
  if (pre.includes('~') || pre.includes('^') || pre.includes('\\') || pre.includes('\\"')) {
    mathHandler("<ul><li>construction error: <ul> <li>don't include ~, ^, \\ </li> <li>  
variable names can't be empty</li></ul></li></ul>");
    output.focus();
    return;
  }
} catch (Exception) {
  mathHandler("<ul><li>construction error: <ul> <li> don't include ~, ^, \\ </li> <li>  
variable names can't be empty </li> </ul></li></ul>");
  output.focus();
  return;
}
if (testContext(pre)) {
  mathHandler("<ul><li>construction error: <ul> <li> don't repeat variable names in the  
context </li> </ul></li></ul>");
  output.focus();
  return;
}
var response="";
xhr.onload = () => {
  if (xhr.status >= 200 && xhr.status < 300) {
    response = xhr.responseText;
    console.log("response=", response);
    console.log(JSON.parse(response));
    var res = toHTML(JSON.parse(response));
    mathHandler(res);
  }
};
xhr.open('POST', 'http://localhost:8000');
xhr.setRequestHeader('Content-Type', 'application/json');

```

```

xhr.send(pre);
output.focus();
}
function mathHandler(res) {
  const math = document.querySelector('#display');
  math.innerHTML = res;
  return math;
}

var options = {
  grid: {
    spacing: 25,
    length: 3,
    colour: '#ccc'
  },
  move: {
    scrollbars: true,
    drag: true,
    wheel: true,
  },
  zoom: {
    controls: true,
    startScale: 1.0,
    maxScale: 4,
    minScale: 0.25,
    scaleSpeed: 1.1
  },
  toolbox: document.getElementById('toolbox')
}

</script>

<xml xmlns="https://developers.google.com/blockly/xml" id="workspace—blocks"
  style="display: none">
  <block type="set_term" x="-1000" y="100"></block>
  <block type="set_context" x="-1000" y="200"></block>
</xml>
</body>
</html>

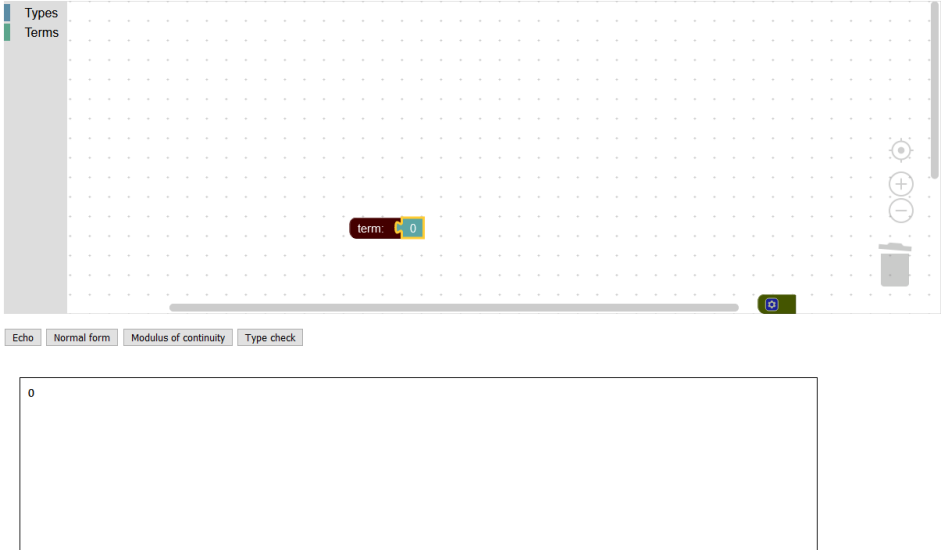
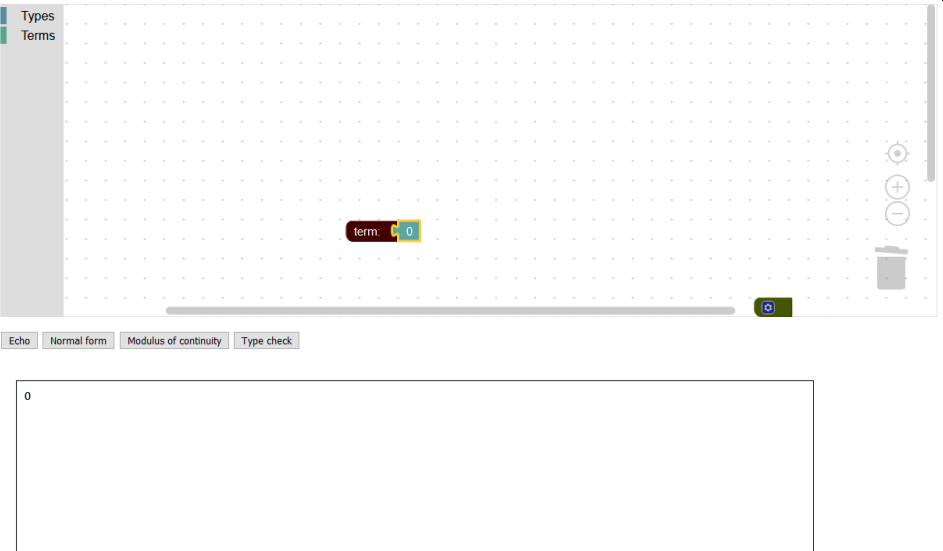
```

# Appendix B

## Evidence of tests


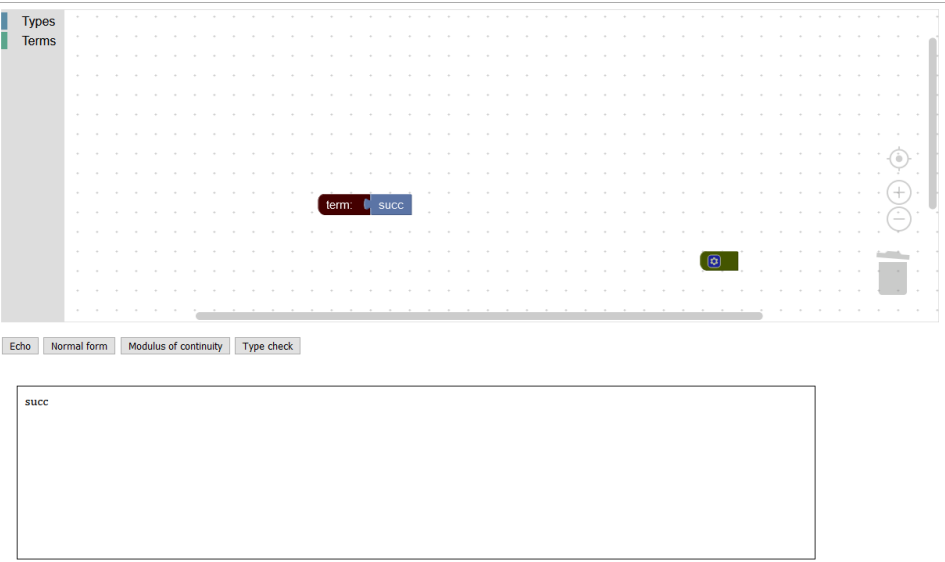
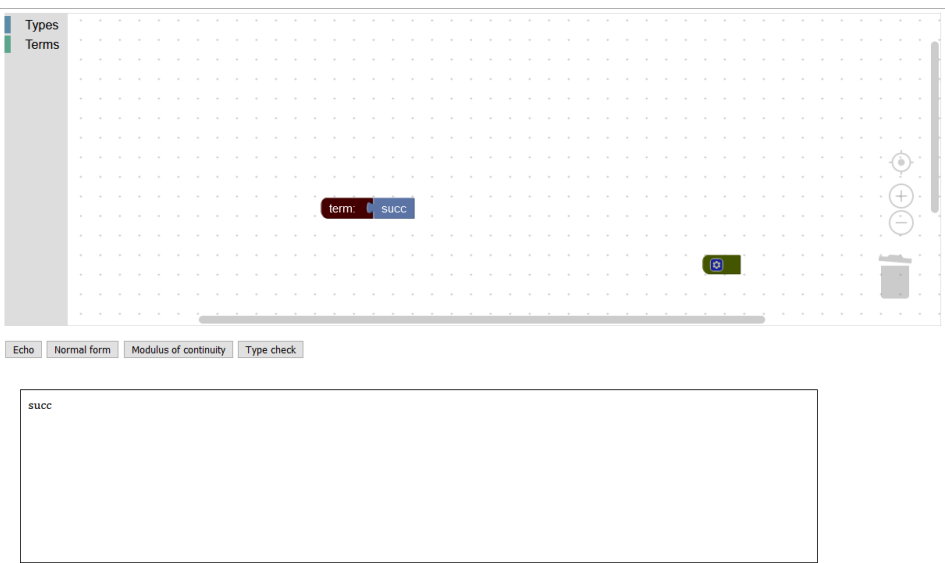
Evidence of testing is presented below,

Table B.1: Evidence of testing

Index	Evidence
1	
2	

Continued on next page

Table B.1 – continued from previous page

Index	Evidence
3	
4	
5	

Continued on next page

Table B.1 – continued from previous page

Index	Evidence
6	<div><div><div>Types</div><div>Terms</div></div><div><div>term: succ</div><div></div></div><div><div>Echo</div><div>Normal form</div><div>Modulus of continuity</div><div>Type check</div></div><div><div><math>N \rightarrow N</math></div></div></div>
7	<div><div><div>Types</div><div>Terms</div></div><div><div>term: rec N</div><div></div></div><div><div>Echo</div><div>Normal form</div><div>Modulus of continuity</div><div>Type check</div></div><div><div><math>rec_N</math></div></div></div>
8	<div><div><div>Types</div><div>Terms</div></div><div><div>term: rec N</div><div></div></div><div><div>Echo</div><div>Normal form</div><div>Modulus of continuity</div><div>Type check</div></div><div><div><math>rec_N</math></div></div></div>

Continued on next page

Table B.1 – continued from previous page

Index	Evidence
9	 <p>term: rec : N</p> <p><math>N \rightarrow ((N \rightarrow (N \rightarrow N)) \rightarrow (N \rightarrow N))</math></p>
10	 <p>term: a</p> <p>a</p>
11	 <p>term: a</p> <p>a</p>

Continued on next page

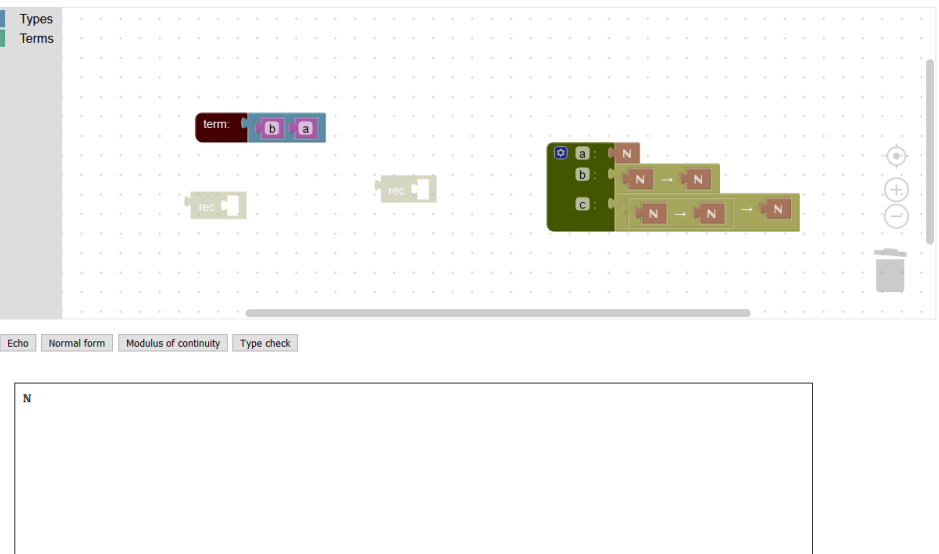
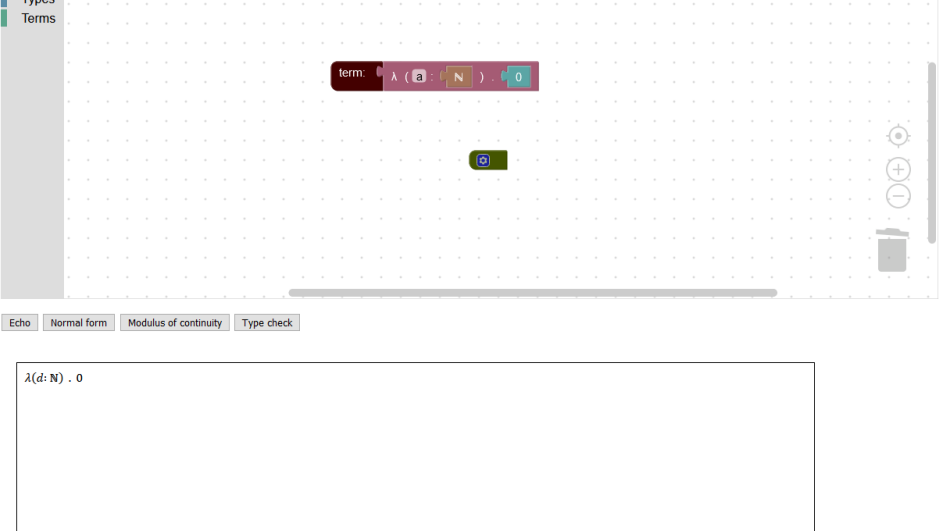
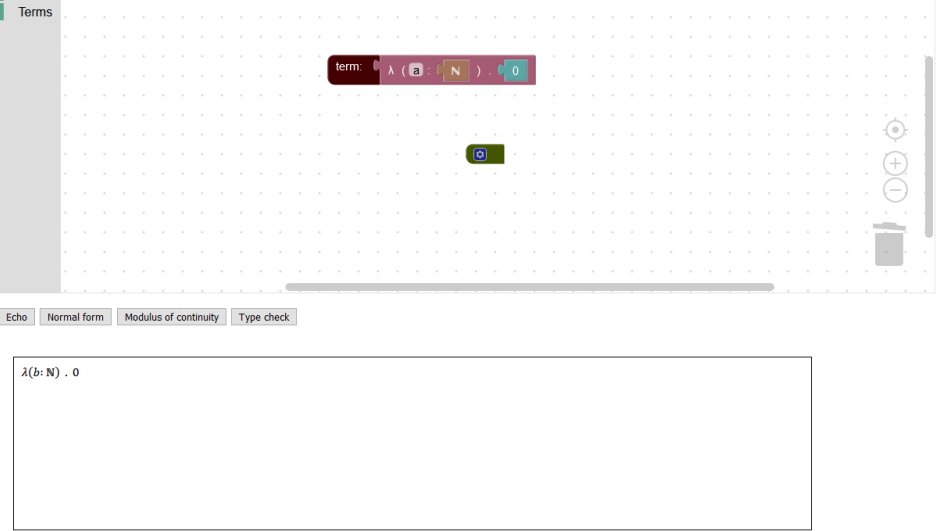


Table B.1 – continued from previous page

Index	Evidence
12	 <p>term: a</p> <p>Types Terms</p> <p>a : N b : N → N c : N → N → N</p> <p>Echo Normal form Modulus of continuity Type check</p> <p>N</p>
13	 <p>term: b a</p> <p>Types Terms</p> <p>a : N b : N → N c : N → N → N</p> <p>Echo Normal form Modulus of continuity Type check</p> <p>b a</p>
14	 <p>term: b a</p> <p>Types Terms</p> <p>a : N b : N → N c : N → N → N</p> <p>Echo Normal form Modulus of continuity Type check</p> <p>b a</p>

Continued on next page

Table B.1 – continued from previous page

Index	Evidence
15	 <p>term: <math>b</math> <math>a</math></p> <p>rec: <math>rec</math> <math>rec</math></p> <p><math>a : N</math>  <math>b : N \rightarrow N</math>  <math>c : N \rightarrow N \rightarrow N</math></p> <p>Echo Normal form Modulus of continuity Type check</p> <p><math>N</math></p>
16	 <p>term: <math>\lambda (a : N) . 0</math> <math>0</math></p> <p>Echo Normal form Modulus of continuity Type check</p> <p><math>\lambda (d : N) . 0</math></p>
17	 <p>term: <math>\lambda (a : N) . 0</math> <math>0</math></p> <p>Echo Normal form Modulus of continuity Type check</p> <p><math>\lambda (b : N) . 0</math></p>

Continued on next page

Table B.1 – continued from previous page

Index	Evidence
18	<div data-bbox="375 264 1321 613"> <div> Types Terms </div> <div> term: <math>\lambda (x : N) . 0</math> </div> <div> </div> <div> Echo Normal form Modulus of continuity Type check </div> <div> <math>N \rightarrow N</math> </div> </div>
19	<div data-bbox="375 837 1321 1187"> <div> Types Terms </div> <div> term: <math>\text{rec } (x : N \rightarrow N) . x</math> </div> <div> </div> <div> Echo Normal form Modulus of continuity Type check </div> <div> <math>\text{rec}_{N \rightarrow N}</math> </div> </div>
20	<div data-bbox="375 1404 1321 1753"> <div> Types Terms </div> <div> term: <math>\text{rec } (x : N \rightarrow N) . x</math> </div> <div> </div> <div> Echo Normal form Modulus of continuity Type check </div> <div> <math>\text{rec}_{N \rightarrow N}</math> </div> </div>

Continued on next page

Table B.1 – continued from previous page

Index	Evidence
21	 <p>Types Terms</p> <p>term: rec</p> <p><math>(N \rightarrow N) \rightarrow ((N \rightarrow ((N \rightarrow N) \rightarrow (N \rightarrow N))) \rightarrow (N \rightarrow (N \rightarrow N)))</math></p> <p>Echo Normal form Modulus of continuity Type check</p>
22	 <p>Types Terms</p> <p>term: <math>\lambda (a : N) . a</math></p> <p>0</p> <p>Echo Normal form Modulus of continuity Type check</p> <p>0</p>
23	 <p>Types Terms</p> <p>term: c</p> <p><math>M_{db} (\lambda (d : Nb) . \lambda (e : N \rightarrow N) . e (Vd e) : \lambda (f : N \rightarrow N) . \max (M d f \text{succ} (Vd f)))</math></p> <p>Echo Normal form Modulus of continuity Type check</p>

Continued on next page

Table B.1 – continued from previous page

Index	Evidence
24	 <p>term: <math>\lambda (a : N \rightarrow N) . \text{succ } 0</math></p> <p><math>\lambda(a: N \rightarrow N) . \text{succ } 0</math></p>
25	 <p>term: <math>\lambda (a : N) . (\text{rec}_N a) (\lambda (c : N) . \text{succ})</math></p> <p><math>\lambda(a: N) . (\text{rec}_N a) (\lambda(c: N) . \text{succ})</math></p>
26	 <p>term: <math>\text{succ} (\text{succ} (\text{succ } 0))</math></p> <p><math>\text{succ} (\text{succ} (\text{succ } 0))</math></p>

Continued on next page

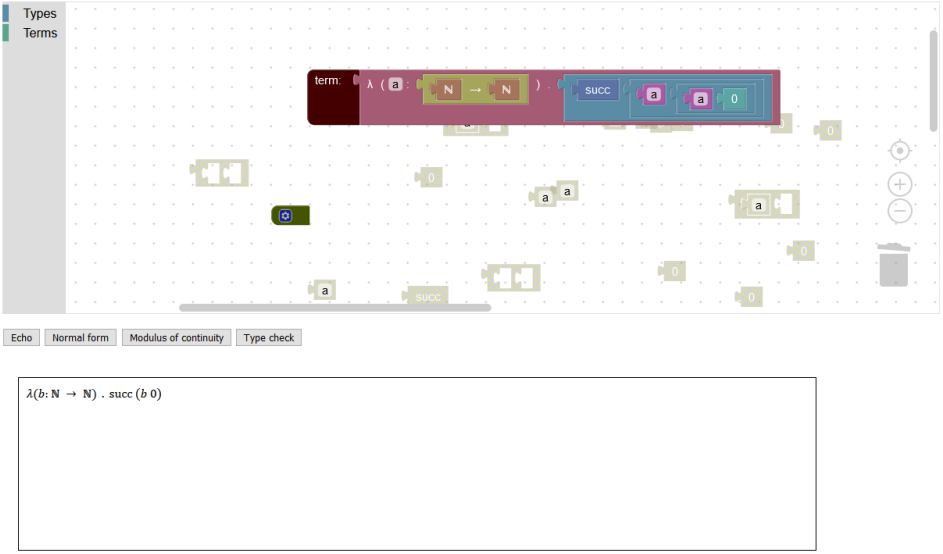
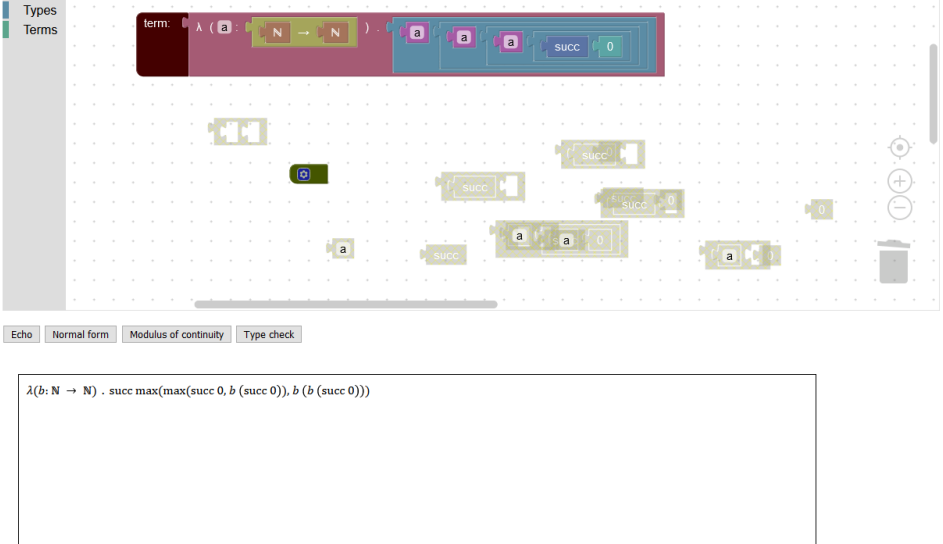
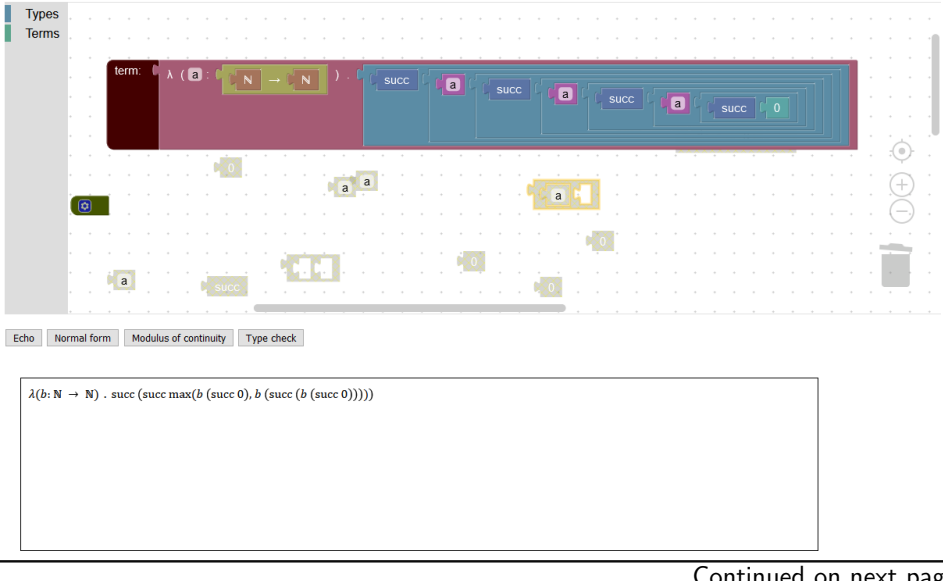


Table B.1 – continued from previous page

Index	Evidence
30	<div data-bbox="375 262 1321 577"> </div> <div data-bbox="375 584 678 607"> Echo   Normal form   Modulus of continuity   Type check </div> <div data-bbox="395 636 1193 808"> <pre>succ (succ (succ (succ (succ (succ (succ (succ 0)))))))</pre> </div>
31	<div data-bbox="375 826 1321 1142"> </div> <div data-bbox="375 1149 678 1171"> Echo   Normal form   Modulus of continuity   Type check </div> <div data-bbox="395 1200 1193 1373"> <pre>succ (succ (succ (succ (succ (succ (succ (succ 0)))))))</pre> </div>
32	<div data-bbox="375 1391 1321 1706"> </div> <div data-bbox="375 1713 678 1736"> Echo   Normal form   Modulus of continuity   Type check </div> <div data-bbox="395 1765 1193 1937"> <pre>succ 0</pre> </div>

Continued on next page

Table B.1 – continued from previous page

Index	Evidence
33	 <p>term: <math>\lambda (a : 0 \rightarrow N) . \text{succ} (a a 0)</math></p> <p>Echo Normal form Modulus of continuity Type check</p> <p><math>\lambda(b: N \rightarrow N) . \text{succ} (b 0)</math></p>
34	 <p>term: <math>\lambda (a : 0 \rightarrow N) . \text{succ} \max(\text{succ } 0, b (\text{succ } 0))</math></p> <p>Echo Normal form Modulus of continuity Type check</p> <p><math>\lambda(b: N \rightarrow N) . \text{succ} \max(\max(\text{succ } 0, b (\text{succ } 0)), b (b (\text{succ } 0)))</math></p>
35	 <p>term: <math>\lambda (a : 0 \rightarrow N) . \text{succ} (\text{succ} \max(b (\text{succ } 0), b (\text{succ} (b (\text{succ } 0))))</math></p> <p>Echo Normal form Modulus of continuity Type check</p> <p><math>\lambda(b: N \rightarrow N) . \text{succ} (\text{succ} \max(b (\text{succ } 0), b (\text{succ} (b (\text{succ } 0)))))</math></p>

Continued on next page

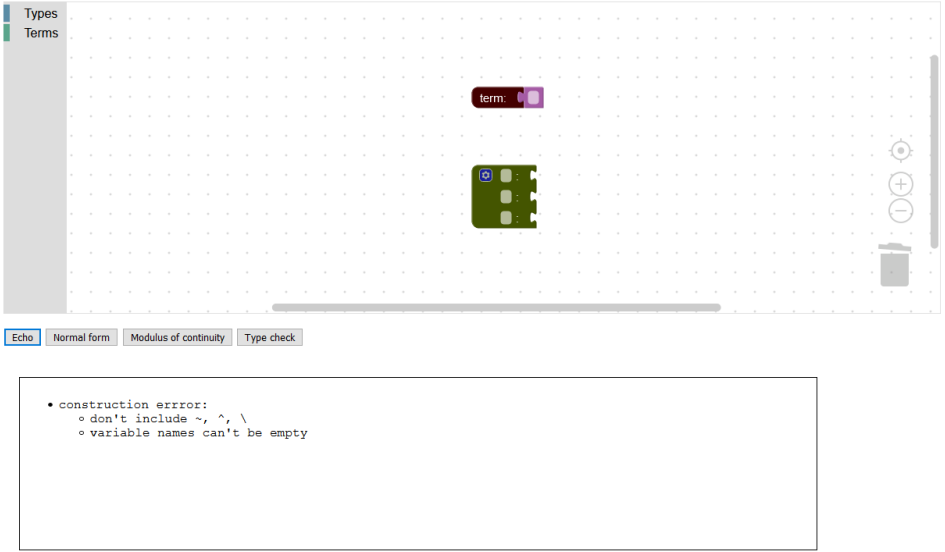
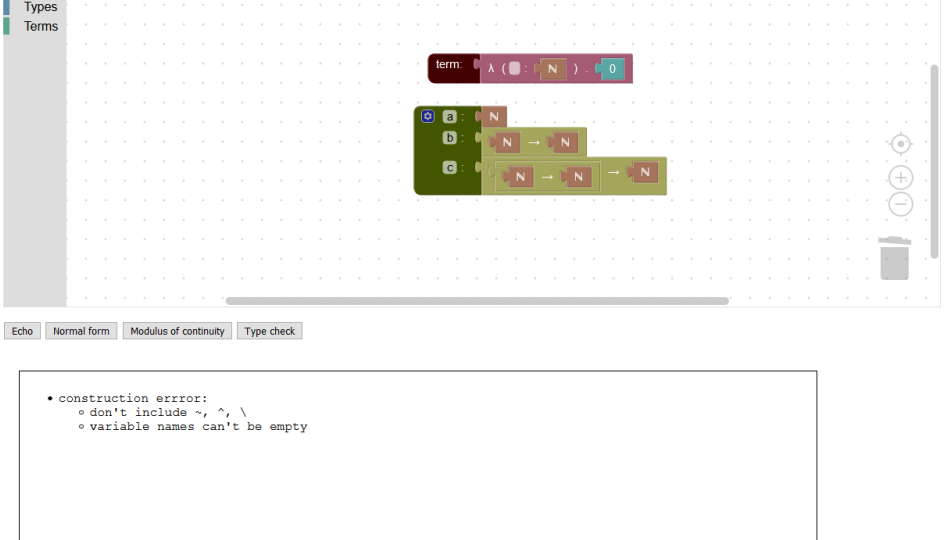
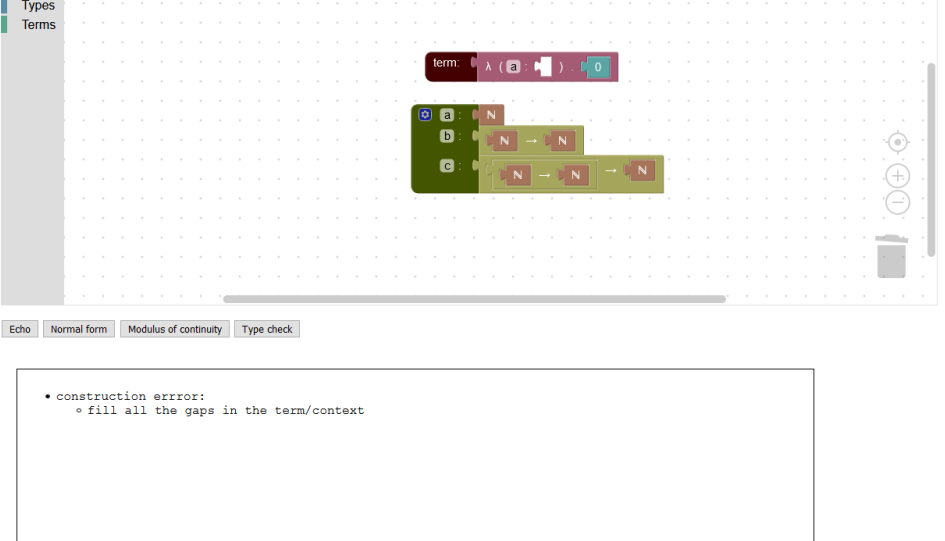


Table B.1 – continued from previous page

Index	Evidence
36	<div data-bbox="375 264 1321 577"> </div> <div data-bbox="395 638 1193 815"> <math display="block">\lambda(b: N \rightarrow N) . \text{succ max}(\text{max}(b \ 0, \text{succ}(b \ (b \ 0))), b \ (\text{succ}(b \ (b \ 0))))</math> </div>
37	<div data-bbox="375 833 1321 1146"> </div> <div data-bbox="395 1207 1193 1384"> <math display="block">\lambda(b: N \rightarrow N) . \text{succ}(\text{succ}(\text{succ}(b \ (\text{succ}(\text{succ } 0)))))</math> </div>
101	<div data-bbox="375 1402 1321 1715"> </div> <div data-bbox="395 1776 1193 1953"> <ul style="list-style-type: none"> <li>• construction error:             <ul style="list-style-type: none"> <li>◦ fill all the gaps in the term/context</li> </ul> </li> </ul> </div>

Continued on next page

Table B.1 – continued from previous page

Index	Evidence
102	
103	
104	

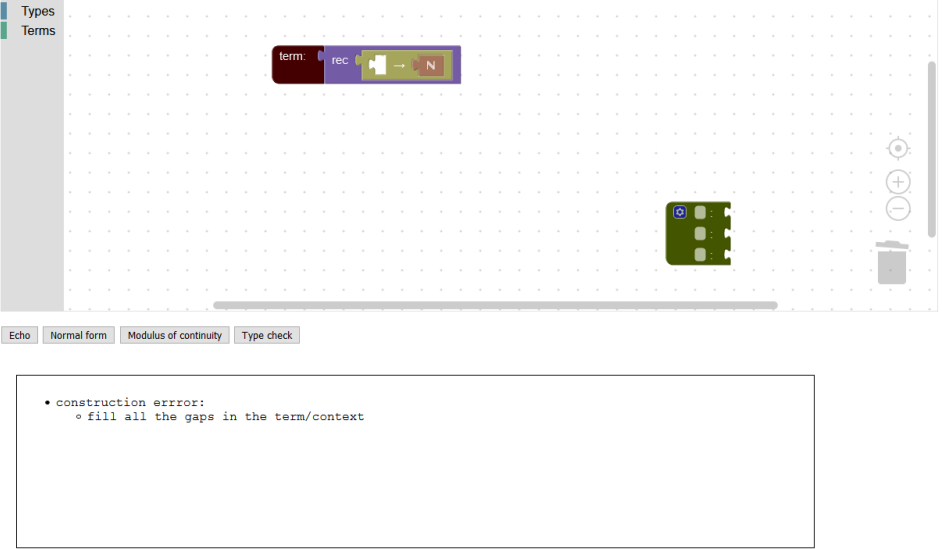
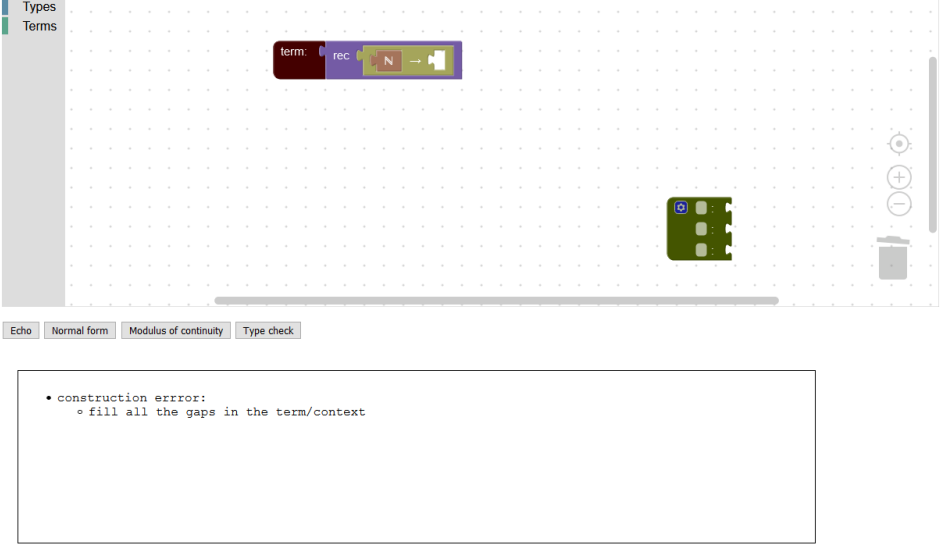
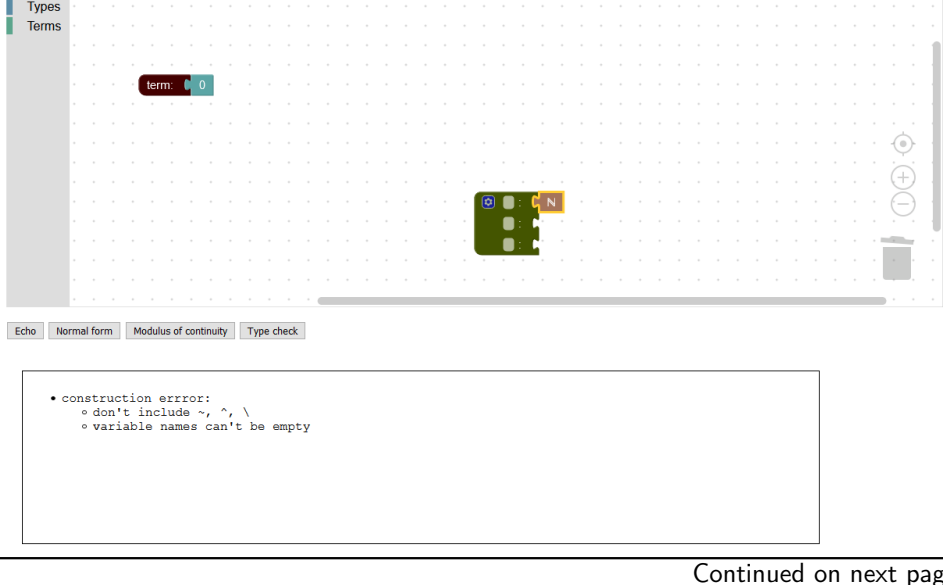
Continued on next page

Table B.1 – continued from previous page

Index	Evidence
105	<div data-bbox="373 264 1318 613"> <div> Types Terms </div> <div>  </div> <div data-bbox="389 645 1193 815"> <ul style="list-style-type: none"> <li>• construction error: <ul style="list-style-type: none"> <li>◦ fill all the gaps in the term/context</li> </ul> </li> </ul> </div> </div>
106	<div data-bbox="373 837 1318 1187"> <div> Types Terms </div> <div>  </div> <div data-bbox="389 1218 1193 1388"> <ul style="list-style-type: none"> <li>• construction error: <ul style="list-style-type: none"> <li>◦ fill all the gaps in the term/context</li> </ul> </li> </ul> </div> </div>
107	<div data-bbox="373 1415 1318 1765"> <div> Types Terms </div> <div>  </div> <div data-bbox="389 1796 1193 1966"> <ul style="list-style-type: none"> <li>• construction error: <ul style="list-style-type: none"> <li>◦ fill all the gaps in the term/context</li> </ul> </li> </ul> </div> </div>

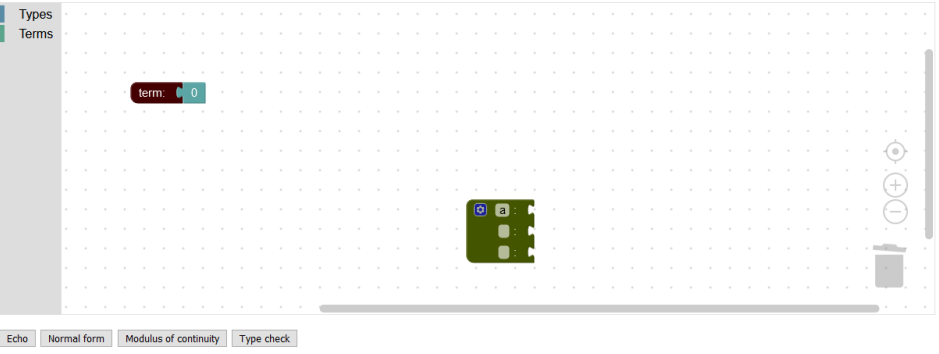
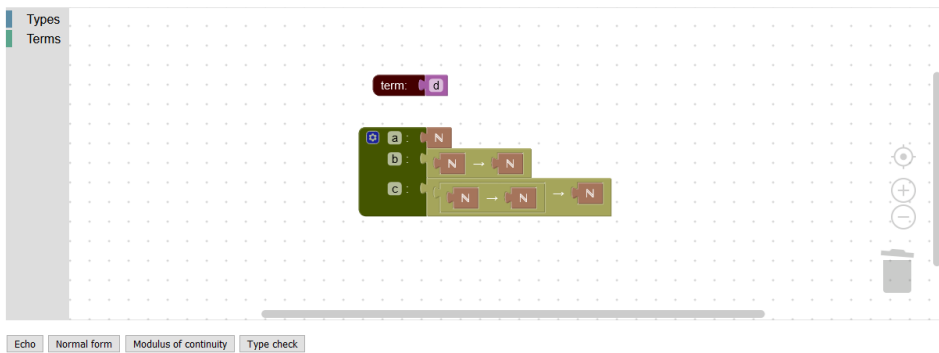
Continued on next page

Table B.1 – continued from previous page

Index	Evidence
108	
109	
110	

Continued on next page


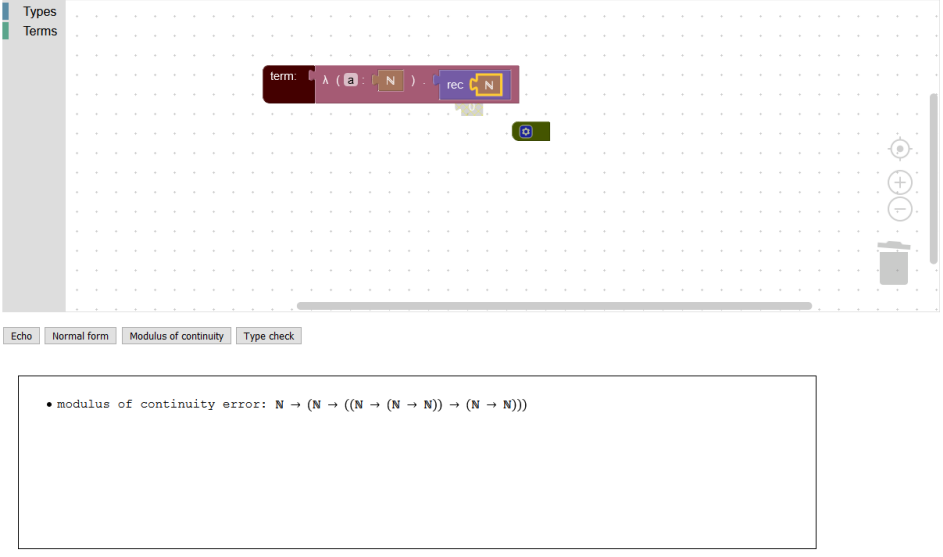
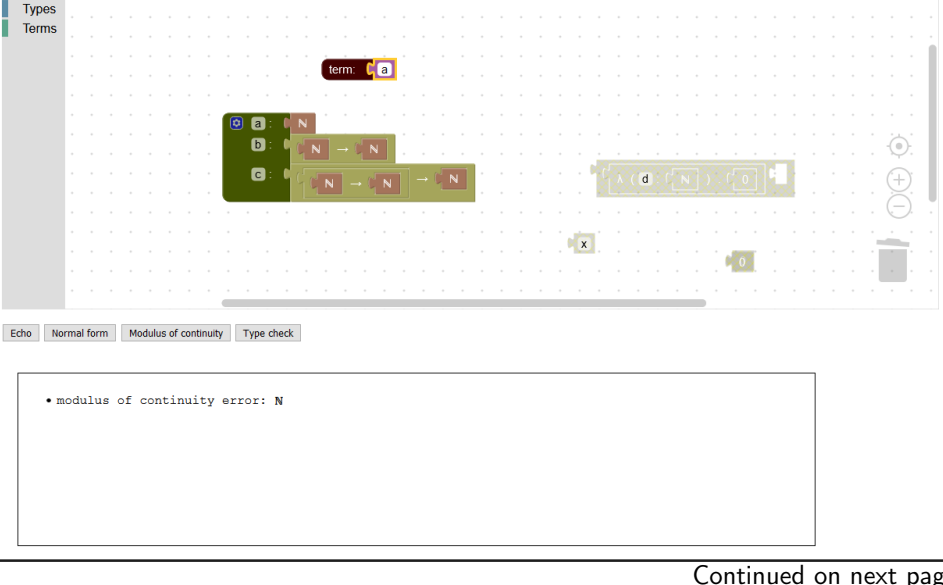
Table B.1 – continued from previous page

Index	Evidence
111	 <p>term: 0</p> <p>Echo Normal form Modulus of continuity Type check</p> <ul style="list-style-type: none"> <li>• construction error:       <ul style="list-style-type: none"> <li>◦ don't include ~, ^, \</li> <li>◦ variable names can't be empty</li> </ul> </li> </ul>
112	 <p>term: d</p> <p>Echo Normal form Modulus of continuity Type check</p> <ul style="list-style-type: none"> <li>• var not found error: d</li> </ul>
113	 <p>term: 0 succ</p> <p>Echo Normal form Modulus of continuity Type check</p> <ul style="list-style-type: none"> <li>• application error: 0: N, succ: N → N</li> </ul>

Continued on next page



Table B.1 – continued from previous page

Index	Evidence
117	
118	
119	

Continued on next page

Table B.1 – continued from previous page

Index	Evidence
120	<div data-bbox="375 264 1321 571"> </div>
121	<div data-bbox="375 824 1321 1131"> </div>
122	<div data-bbox="375 1384 1321 1691"> </div> <div data-bbox="395 1765 1193 1935"> <p>• modulus of continuity error: <math>N \rightarrow ((N \rightarrow (N \rightarrow N)) \rightarrow (N \rightarrow N))</math></p> </div>



## Appendix C

# Impact of COVID-19

One difficulty has been the lack of a whiteboard to ask my supervisor questions, which has been difficult for a visual thinker like me (I think that this project makes it quite clear that I am a visual thinker). At one meeting I struggled to ask my supervisor a mathematical question, and at a different meeting, I had a similar struggle in asking my supervisor a programming question. If I were in a physical meeting, I would have used a whiteboard to explain the questions. In other meetings I overcame this by putting the idea somewhere in to my document or my code, so I could have something to refer to and in one case connecting a graphics tablet and screen-sharing drawings drawn using it, but using a graphics tablet can be slightly time-consuming.

I also started to have eye strain from having to use my laptop so much.



**Department of Computer Science**  
**12-Point Ethics Checklist for UG and MSc Projects**

**Student** Kieran Maharaj

**Academic Year  
or Project Title** 2020/2021

**Supervisor** Thomas Powell

*Does your project involve people for the collection of data other than you and your supervisor(s)?*

YES / **NO**

If the answer to the previous question is YES, you need to answer the following questions, otherwise you can ignore them.

This document describes the 12 issues that need to be considered carefully before students or staff involve other people ('participants' or 'volunteers') for the collection of information as part of their project or research. Replace the text beneath each question with a statement of how you address the issue in your project.

1. *Will you prepare a Participant Information Sheet for volunteers?*

YES / NO

This means telling someone enough in advance so that they can understand what is involved and why – it is what makes informed consent informed.

2. *Will the participants be informed that they could withdraw at any time?*

YES / NO

All participants have the right to withdraw at any time during the investigation, and to withdraw their data up to the point at which it is anonymised. They should be told this in the briefing script.

3. *Will there be any intentional deception of the participants?*

YES / NO

Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.

4. *Will participants be de-briefed?*

YES / NO

The investigator must provide the participants with sufficient information in the debriefing to enable them to understand the nature

of the investigation. This phase might wait until after the study is completed where this is necessary to protect the integrity of the study.

5. ***Will participants voluntarily give informed consent?*** YES / NO

Participants MUST consent before taking part in the study, informed by the briefing sheet. Participants should give their consent explicitly and in a form that is persistent –e.g. signing a form or sending an email. Signed consent forms should be kept by the supervisor after the study is complete. If your data collection is entirely anonymous and does not include collection of personal data you do not need to collect a signature. Instead, you should include a checkbox, which must be checked by the participant to indicate that informed consent has been given.
6. ***Will the participants be exposed to any risks greater than those encountered in their normal work life (e.g., through the use of non-standard equipment)?*** YES / NO

Investigators have a responsibility to protect participants from physical and mental harm during the investigation. The risk of harm must be no greater than in ordinary life.
7. ***Will you be offering any incentive to the participants?*** YES / NO

The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.
8. ***Will you be in a position of authority or influence over any of your participants?*** YES / NO

A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any experiment.
9. ***Will any of your participants be under the age of 16?*** YES / NO

Parental consent is required for participants under the age of 16.
10. ***Will any of your participants have an impairment that will limit Their understanding or communication?*** YES / NO

Additional consent is required for participants with impairments.
11. ***Will the participants be informed of your contact details?*** YES / NO

All participants must be able to contact the investigator after the investigation. They should be given the details of the Supervisor as part of the debriefing.

12. *Will you have a data management plan for all recorded data?* YES / NO

Personal data is anything which could be used to identify a person, or which can be related to an identifiable person. All personal data (hard copy and/or soft copy) should be anonymized (with the exception of consent forms) and stored securely on university servers (not the cloud).