Fundamentals of Programming Languages
Assignment 1
Statics and Dynamics

Mestrado em (Engenharia) Informática
Faculdade de Ciências da Universidade de Lisboa

2022/2023

1 Natural numbers and induction (20%)

We have introduced natural numbers by means of a grammar such as the one below.

$\mathtt{a} ::=$		*naturals:*
	$\mathtt{0}$	*constant zero*
	$\mathtt{succ(a)}$	*successor*

An alternative presentation uses rules featuring judgements of the form $\mathtt{a\ nat}$. The rules are as follows.

$$\text{N-ZERO}\ \frac{}{\mathtt{0\ nat}} \qquad \text{N-SUCC}\ \frac{\mathtt{a\ nat}}{\mathtt{succ(a)\ nat}}$$

A. Show that $\mathtt{succ(succ(succ(succ(succ(0)))))\ nat}$ by exhibiting an appropriate derivation.

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\mathtt{0\ nat}}{\mathtt{succ(0)\ nat}}\ \text{N-ZERO}}{\mathtt{succ(succ(0))\ nat}}\ \text{N-SUCC}}{\mathtt{succ(succ(succ(0)))\ nat}}\ \text{N-SUCC}}{\mathtt{succ(succ(succ(succ(0))))\ nat}}\ \text{N-SUCC}}{\mathtt{succ(succ(succ(succ(succ(0)))))\ nat}}\ \text{N-SUCC}$$

The term is well-typed by definition, since every subterm is well-typed by definition, so the typing statement $\mathtt{succ(succ(succ(succ(succ(0)))))\ nat}$ is confirmed to be true.

Now consider the even and the odd predicate, also defined in rule format

$$\text{E-ZERO}\ \frac{}{\mathsf{0\ even}} \qquad \text{E-SUCC}\ \frac{\mathsf{a\ odd}}{\mathtt{succ}(\mathsf{a})\ \mathsf{even}} \qquad \text{O-SUCC}\ \frac{\mathsf{a\ even}}{\mathtt{succ}(\mathsf{a})\ \mathsf{odd}}$$

B. Show that the successor of the successor of an odd number is odd. Then show that the successor of the successor of an odd even number is even.

$$\dfrac{\dfrac{\mathsf{a\ even}}{(\mathtt{succ}(\mathsf{a}))\ \mathsf{odd}}\ \text{O-SUCC}}{\mathtt{succ}(\mathtt{succ}(\mathsf{a}))\ \mathsf{even}}\ \text{E-SUCC} \qquad \dfrac{\dfrac{\mathsf{a\ odd}}{(\mathtt{succ}(\mathsf{a}))\ \mathsf{even}}\ \text{E-SUCC}}{\mathtt{succ}(\mathtt{succ}(\mathsf{a}))\ \mathsf{odd}}\ \text{O-SUCC}$$

After terms (natural numbers) and predicates (odd and even), also n-ary relations can be written in rule format. Consider the addition function on natural numbers. There are only two rules. One for the base case (0) and the other for step (succ(a)). The rule for the base case says that

if b is a natural number, then $0 + b = b$.

That for the successor is for you to derive. To say that $a + b = c$ we may use a judgement of the form sum(a, b, c).

C. Define a ternary relation sum(a, b, c) by means of two rules. The relation should be defined on natural numbers only: we do not want to be able to conclude, for example, that sum(succ(0), **fish**, succ(**fish**)).

$$\text{S-SUM}\ \frac{\mathtt{sum}(\mathsf{a},\mathsf{b},\mathsf{c})}{\mathtt{sum}(\mathtt{succ}(\mathsf{a}),\mathsf{b},\mathtt{succ}(\mathsf{c}))} \qquad \text{S-SUMZERO}\ \frac{\mathsf{b\ nat}}{\mathtt{sum}(0,\mathsf{b},\mathsf{b})}$$

D. Using rule induction show that

1. For every a nat and b nat, there is a c nat such that sum(a, b, c);

   This has to be proved by induction on e
   Assuming $c = a + b$

   Case e = succ(a) + b:
   By the rule E-SUCC, it is known that succ(a) nat, and we know that the S-SUM relation can only be defined on nat values, so we assume b nat. With this, following the induction hypothesis, if a nat and b nat are true, it concludes that there is a c nat knowing that sum(a, b, c). Furthermore, when applying the rule N-SUCC on c, it is concluded that succ(c) nat because c nat. So, it is true that sum(succ(a), b, succ(c)).

   Case e = 0 + b:
   By the rule E-SUCC, it is known that 0 nat, and we know that the S-SUMZERO relation can only be defined on nat values, so we assume

b nat.
With this, following the induction hypothesis, if a nat and b nat are true, it concludes that there is a c nat knowing that $\mathsf{sum}(\mathsf{a}, \mathsf{b}, \mathsf{c})$.

2. c nat is unique.

   Case $\mathsf{c} = \mathsf{succ}(\mathsf{a})$:
   By the Inversion Typing Lemma in 2.D or by the N-ZERO rule, it is concluded that 0 nat, so we can apply the induction hypothesis and simply conclude that 0 is unique, so c is also unique.

   Case $\mathsf{c} = 0$:
   By the Inversion Typing Lemma in 2.D or by the N-SUCC rule, it is concluded that $\mathsf{succ}(\mathsf{a})$ nat if a nat confirms to be true, so we can apply the induction hypothesis and conclude that if a nat then t is unique, which justifies, in the previously mentioned rule, that $\mathsf{succ}(\mathsf{a})$ nat, in other words, unique, so c is also unique.

Together these results say that sum is a total function.

E. Give an inductive definition of the judgement $\mathtt{max}(\mathtt{a}, \mathtt{b}, \mathtt{c})$ where $\mathtt{a}$ nat, $\mathtt{b}$ nat and $\mathtt{c}$ nat, with the meaning that $\mathtt{c}$ is the largest of $\mathtt{a}$ and $\mathtt{b}$.

[Definition] Terms, Inductively

1. 0 nat $\subseteq \mathtt{a}$
2. if $\mathtt{a} \subseteq \mathtt{a}$, then $\mathtt{succ}(\mathtt{a}) \subseteq \mathtt{a}$
3. if $\mathtt{a} \subseteq \mathtt{a}$ and $\mathtt{b} \subseteq \mathtt{a}$, then $\mathtt{c} \subseteq \mathtt{a}$, since $\mathtt{c} = \mathtt{a} \vee \mathtt{c} = \mathtt{b}$ (assuming $\mathtt{max}(\mathtt{a}, \mathtt{b}, \mathtt{c})$)

Considering a the set of Terms defined in beginning of the report as a metavariable.
[Definition] Terms, by Inference Rules

$$\text{S-MAX}\ \frac{\mathtt{max}(\mathtt{a}, \mathtt{b}, \mathtt{c})}{\mathtt{max}(\mathtt{succ}(\mathtt{a}), \mathtt{succ}(\mathtt{b}), \mathtt{succ}(\mathtt{c}))} \qquad \text{S-MAXZERO}\ \frac{\mathtt{a}\ \mathsf{nat}}{\mathtt{max}(\mathtt{a}, 0, \mathtt{a})} \qquad \text{S-MAXZERO2}\ \frac{\mathtt{b}\ \mathsf{nat}}{\mathtt{max}(0, \mathtt{b}, \mathtt{b})}$$

2 Natural numbers, strings and typing (20%)

We now define a language of natural numbers, strings and operations on these.

e ::=	*terms:*
0	*constant zero*
succ(e)	*successor*
e + e	*addition*
ε	*empty string*
Ae	*non-empty string*
Be	*non-empty string*
len(e)	*string length*

Natural numbers and their only operation (+) are as defined in Section 1. Strings are built from ε, the empty string, and by prefixing a given string by letters A and B. For simplicity strings are built from letters A and B alone. Examples of strings are ε, Aε, Bε, BABAε and ABBABABAABABABABε. The length of a string, len, denotes a natural number. For example, len(BABAε) denotes the natural number succ(succ(succ(succ(0)))).

A. Suggest a grammar for types appropriate to type terms. Use metavariable T to denote arbitrary types.

$$T ::= Nat | String$$

B. Present a type system assigning types to terms. Use judgements of the from e : T.

$$\text{T-ZERO}\quad \frac{}{\mathtt{0} : \mathsf{Nat}} \qquad \text{T-SUCC}\quad \frac{\mathtt{e} : \mathsf{Nat}}{\mathtt{succ(e)} : \mathsf{Nat}} \qquad \text{T-SUM}\quad \frac{\mathtt{e1} : \mathsf{Nat} \qquad \mathtt{e2} : \mathsf{Nat}}{\mathtt{e1} + \mathtt{e2} : \mathsf{Nat}}$$

$$\text{T-EMPTY}\quad \frac{}{\varepsilon : \mathsf{String}} \qquad \text{T-ASTRING}\quad \frac{\mathtt{e} : \mathsf{String}}{\mathtt{Ae} : \mathsf{String}} \qquad \text{T-BSTRING}\quad \frac{\mathtt{e} : \mathsf{String}}{\mathtt{Be} : \mathsf{String}} \qquad \text{T-LEN}\quad \frac{\mathtt{e} : \mathsf{String}}{\mathtt{len(e)} : \mathsf{Nat}}$$

C. Exhibit a term that is typable according to the type system just defined and another that is untypable. Show that the terms you have selected are indeed typable and untypable.

[Definition] A term t is typable (or well typed) if there is some T such that $t : T$.

The following term is typable:

$$
\begin{prooftree}
\AxiomC{}
\LeftLabel{T-ZERO}
\UnaryInfC{$\mathtt{0} : \mathsf{Nat}$}
\LeftLabel{T-SUCC}
\UnaryInfC{$\mathtt{succ(0)} : \mathsf{Nat}$}
\LeftLabel{T-SUCC}
\UnaryInfC{$\mathtt{succ(succ(0))} : \mathsf{Nat}$}
\AxiomC{}
\RightLabel{T-EMPTY}
\UnaryInfC{$\varepsilon : \mathsf{String}$}
\RightLabel{T-BSTRING}
\UnaryInfC{$\mathtt{B}\varepsilon : \mathsf{String}$}
\RightLabel{T-BSTRING}
\UnaryInfC{$\mathtt{BB}\varepsilon : \mathsf{String}$}
\RightLabel{T-ASTRING}
\UnaryInfC{$\mathtt{ABB}\varepsilon : \mathsf{String}$}
\RightLabel{T-LEN}
\UnaryInfC{$\mathtt{len(ABB}\varepsilon\mathtt{)} : \mathsf{Nat}$}
\RightLabel{T-SUCC}
\UnaryInfC{$\mathtt{succ(len(ABB}\varepsilon\mathtt{))} : \mathsf{Nat}$}
\LeftLabel{T-PLUS}
\BinaryInfC{$\mathtt{succ(succ(0))} + \mathtt{succ(len(ABB}\varepsilon\mathtt{))} : \mathsf{Nat}$}
\AxiomC{}
\RightLabel{T-ZERO}
\UnaryInfC{$\mathtt{0} : \mathsf{Nat}$}
\RightLabel{T-SUCC}
\UnaryInfC{$\mathtt{succ(0)} : \mathsf{Nat}$}
\RightLabel{T-SUCC}
\UnaryInfC{$\mathtt{succ(succ(0))} : \mathsf{Nat}$}
\RightLabel{T-PLUS}
\BinaryInfC{$(\mathtt{succ(succ(0))} + \mathtt{succ(len(ABB}\varepsilon\mathtt{)))} + (\mathtt{succ(succ(0))} : \mathsf{Nat}$}
\end{prooftree}
$$

This is typable because for every instance of the typing rules at the derivation tree, each pair (t, T) in the typing relation is justified by a typing derivation with conclusion $t : T$.

The following term is untypable:

$$
\begin{prooftree}
\AxiomC{}
\LeftLabel{T-EMPTY}
\UnaryInfC{$\varepsilon : \mathsf{String}$}
\LeftLabel{T-LEN}
\UnaryInfC{$\mathtt{len(}\varepsilon\mathtt{)} : \mathsf{Nat}$}
\LeftLabel{T-SUCC}
\UnaryInfC{$\mathtt{succ(len(}\varepsilon\mathtt{))} : \mathsf{Nat}$}
\AxiomC{}
\RightLabel{T-??}
\UnaryInfC{$\mathtt{A}\varepsilon : \mathsf{Nat}$}
\RightLabel{T-PLUS}
\BinaryInfC{$\mathtt{succ(len(}\varepsilon\mathtt{))} + \mathtt{A}\varepsilon : \mathsf{Nat}$}
\end{prooftree}
$$

Similarly to the previous justification, this is not typable because it is not true that for every instance of the typing rules at the derivation tree, each pair (t, T) in the typing relation is justified by a typing derivation with conclusion $t : T$. For example, on the first derivation, the term $A\varepsilon$ doesn't apply to any rule, since $A\varepsilon$ is of type String but it appears as Nat, which is wrong.

D. Show the Unicity of Typing lemma: For every term `e` there is at most one type `T` such that `e` : `T`.

[Lemma] Inversion of the Typing Relation

1. If `0` : R, then R = Nat
2. If `succ e` : R, then R = Nat and `e` : Nat
3. If `e1` + `e2` : R, then R = Nat and `e1` : Nat and `e2` : Nat
4. If ε : R, then R = String
5. If `Ae` : R, then R = String and `e` : String
6. If `Be` : R, then R = String and `e` : String
7. If `len(e)` : R, then R = Nat and `e` : String

[Proof] Immediate from the definition of the typing relation.

[Theorem] Uniqueness of Types (the same as Unicity of Typing lemma)

[Proof] Straightforward induction using the appropriate clause of the inversion lemma, plus the induction hypothesis, for each case.
Being typable = `t` : T form.

1. For `e` $=$ `0`, it's immediately conclusive from the Inversion of the Typing Relation's clause (1). By the induction hypothesis, if `e` is typable, i.e. `e` : Nat, then e is unique. (See T-ZERO rule)

2. For `e` $=$ `succ e1`, it's immediately conclusive from the Inversion of the Typing Relation's clause (2). By the induction hypothesis, if `e1` is typable, i.e. `e1` : Nat, then $e1$ is unique and by T-SUCC, `e` : Nat, in other words, unique.

3. For `e` $=$ `e1` $+$ `e2`, it's immediately conclusive from Inversion of the Typing Relation's clause (3). By the induction hypothesis, if `e1` is typable, i.e. `e1` : Nat, then $e1$ is unique and `e2` is typable, i.e. `e2` : Nat, then $e2$ is unique. By the rule T-SUM, `e` : Nat, in other words, unique.

4. For `e` $= \varepsilon$, it's immediately conclusive from the Inversion of the Typing Relation's clause (4). By the induction hypothesis, if `e` is typable, i.e. `e` : String, then e is unique. (See T-EMPTY rule)

5. For `e` $=$ `Ae1`, it's immediately conclusive from the Inversion of the Typing Relation's clause (5). By the induction hypothesis, if `e1` is typable, i.e. `e1` : String, then $e1$ is unique and by T-ASTRING, `e` : String, in other words, unique.

6. For `e` $=$ `Be1`, it's immediately conclusive from the Inversion of the Typing Relation's clause (6). By the induction hypothesis, if `e1` is typable, i.e. `e1` : String, then $e1$ is unique and by T-BSTRING, `e` : String, in other words, unique.

7. For `e` $=$ `len(e1)`, it's immediately conclusive from the Inversion of the Typing Relation's clause (7). By the induction hypothesis, if `e1` is typable, i.e. `e1` : String, then $e1$ is unique and by T-LEN, `e` : Nat, in other words, unique.

Because of this, we prove that each term t has at most one type.

3 Evaluation and type safety (30%)

The dynamics of the term language is given by a transition system whose states are terms. All states are initial. Final states are *values*. Values form a subset of terms and include the natural numbers and the strings.

A. Define the predicate is-value by means of rules. Use a judgement of the form $\mathsf{e\ val}$.

$$\overset{\text{V-ZERO}}{\frac{}{\mathsf{0\ val}}} \qquad \overset{\text{V-EMPTY}}{\frac{}{\varepsilon\ \mathsf{val}}} \qquad \overset{\text{V-SUCC}}{\frac{\mathsf{e\ val}}{\mathtt{succ(e)}\ \mathsf{val}}} \qquad \overset{\text{V-ASTRING}}{\frac{\mathsf{e\ val}}{\mathtt{Ae}\ \mathsf{val}}} \qquad \overset{\text{V-BSTRING}}{\frac{\mathsf{e\ val}}{\mathsf{Be\ val}}}$$

B. Define a *one-step* evaluation relation that computes the addition and length operations. Use a judgement of the form $\mathsf{e} \to \mathsf{e}$. Take call-by-value for your reduction strategy: first evaluate the parameter(s), and only then apply the operator. For example, if e is a value, then $\mathsf{0} + \mathsf{e}$ should evaluate to e. But if e is *not* a value (and e is typable), then $\mathsf{e} \to \mathsf{e}'$, for some e'. In this case $\mathsf{0} + \mathsf{e}$ should evaluate to $\mathsf{0} + \mathsf{e}'$.

Assuming the existence of the following metavariables:

$$nv ::= 0 \mid Succ\ nv$$
$$sv ::= \varepsilon \mid Asv \mid Bsv$$

$$\overset{\text{E-SUCC}}{\frac{\mathsf{e} \to \mathsf{e}'}{\mathtt{succ(e)} \to \mathtt{succ(e')}}} \qquad \overset{\text{E-SUMZERO}}{\frac{\mathsf{e} \to \mathsf{e}'}{\mathsf{0} + \mathsf{e} \to \mathsf{0} + \mathsf{e}'}} \qquad \overset{\text{E-SUMNV}}{\frac{}{\mathsf{0} + \mathtt{nv} \to \mathtt{nv}}}$$

$$\overset{\text{E-SUMSUCC}}{\frac{}{\mathtt{succ(e1)} + \mathtt{e2} \to \mathtt{succ(e1} + \mathtt{e2)}}} \qquad \overset{\text{E-SUM}}{\frac{\mathtt{e1} \to \mathtt{e1}'}{\mathtt{e1} + \mathtt{e2} \to \mathtt{e1}' + \mathtt{e2}}} \qquad \overset{\text{E-ASTRING}}{\frac{}{\mathtt{Asv} \to \mathtt{sv}}}$$

$$\overset{\text{E-BSTRING}}{\frac{}{\mathtt{Bsv} \to \mathtt{sv}}} \qquad \overset{\text{E-LEN}}{\frac{\mathsf{e} \to \mathsf{e}'}{\mathtt{len(e)} \to \mathtt{succ(len(e'))}}} \qquad \overset{\text{E-LENEMPTY}}{\frac{}{\mathtt{len}(\varepsilon) \to \mathtt{0}}}$$

C. Use your rules to show the following transitions

$$\begin{aligned}
\mathtt{succ(0)} + \mathtt{succ(0)} &\to \mathtt{succ(0} + \mathtt{succ(0))} \\
\mathtt{succ(0} + \mathtt{succ(0))} &\to \mathtt{succ(succ(0))} \\
\mathtt{len(AB}\varepsilon\mathtt{)} &\to \mathtt{succ(len(B}\varepsilon\mathtt{))} \\
\mathtt{succ(len(B}\varepsilon\mathtt{))} &\to \mathtt{succ(succ(len(}\varepsilon\mathtt{)))} \\
\mathtt{len(A}\varepsilon\mathtt{)} + \mathtt{succ(succ(0))} &\to \mathtt{succ(len(A}\varepsilon\mathtt{))} + \mathtt{succ(succ(0))}
\end{aligned}$$

$$\frac{}{\mathtt{succ(0)} + \mathtt{succ(0)} \to \mathtt{succ(0} + \mathtt{succ(0))}}\ \text{E-SUMSUCC}$$

$$\frac{}{\mathtt{succ(0} + \mathtt{succ(0))} \to \mathtt{succ(succ(0))}}\ \text{E-SUMZEROSUCC}$$

$$\dfrac{\dfrac{}{\texttt{AB}\varepsilon \to \texttt{B}\varepsilon}\ \textsc{E-AString}}{\texttt{len(AB}\varepsilon\texttt{)} \to \texttt{succ(len(B}\varepsilon\texttt{))}}\ \textsc{E-Len}$$

$$\dfrac{\dfrac{\dfrac{}{\texttt{B}\varepsilon \to \varepsilon}\ \textsc{E-BString}}{\texttt{len(B}\varepsilon\texttt{)} \to \texttt{succ(len(}\varepsilon\texttt{))}}\ \textsc{E-Len}}{\texttt{succ(len(B}\varepsilon\texttt{))} \to \texttt{succ(succ(len(}\varepsilon\texttt{)))}}\ \textsc{E-Succ}$$

$$\dfrac{\dfrac{\dfrac{}{\texttt{A}\varepsilon \to \varepsilon}\ \textsc{E-AString}}{\texttt{len(A}\varepsilon\texttt{)} \to \texttt{succ(len(}\varepsilon\texttt{))}}\ \textsc{E-Len}}{\texttt{len(A}\varepsilon\texttt{)} + \texttt{succ(succ(0))} \to \texttt{succ(len(}\varepsilon\texttt{))} + \texttt{succ(succ(0))}}\ \textsc{E-Sum}$$

D. Show that if `e` : `T` and no evaluation rule applies to `e`, then `e` val.

[Definition] A term `t` is in normal form if no evaluation rule applies to it, in other words, if there is no `t′` such that `t` → `t′`.
Assuming that no evaluation rule applies to `e`, then this means that `e` is in normal form.

[Theorem] If `t` is in normal form, then `t` is a value.
Since `e` is in normal form, this means that `e` is a value, in other words, `e val`.

E. Show the Progress theorem. If `e` : `T`, then either `e` val or there exists `e′` such that `e` → `e′`.

[Proof] By induction on a derivation of $t : T$. The T-ZERO, T-EMPTY, T-ASTRING and T-BSTRING cases are immediate, since t in these cases is a value. For the other cases, we argue as follows.

Case T-SUCC:
`e` = `succ e1` and `e1` : Nat
By induction hypothesis, either $e1$ is a value or else there is some $e1'$ such that `e1` → `e1′`. If $e1$ is a value, then the canonical forms lemma assures us that it must be a numeric value, in which case so is e. On the other hand, if `e1` → `e1′`, then, by E-SUCC, `succ e1` → `succ e1′`.

Case T-SUM:
`e` = `e1 + e2` and `e1` : Nat and `e2` : Nat
By induction hypothesis, either $e1$ is a value or else there is some $e1'$ such that `e1` → `e1′`. If $e1$ is a value, then the canonical forms lemma assures us that it must be a numeric value, in which there are two possible continuations, 1) either apply E-SUMNV or E-SUMSUCC to e and assume $e2$ to be a value, then

the canonical forms lemma assures us that it must be a numeric value; 2) apply E-SUMZERO to e and assume $e2$ is not a value; On the other hand, if $\texttt{e1} \to \texttt{e1}'$, then, by E-SUM, $\texttt{e} \to \texttt{e} = \texttt{e1}'$ + e2.

Case T-SUCC:
$\texttt{e} = \texttt{len e1}$ and $\texttt{e1} : \mathsf{Nat}$
By induction hypothesis, either $e1$ is a value or else there is some $e1'$ such that $\texttt{e1} \to \texttt{e1}'$. If $e1$ is a value, then the canonical forms lemma assures us that it must be a numeric value, in which case so is e. On the other hand, if $\texttt{e1} \to \texttt{e1}'$, then, by E-LEN, $\texttt{len e1} \to \texttt{succ}(\texttt{len}\ (\texttt{e1}'))$.

F. Show the Preservation theorem. If $\texttt{e} : \texttt{T}$ and $\texttt{e} \to \texttt{e}'$, then $\texttt{e}' : \texttt{T}$.

[Proof] By induction on a derivation of $t : T$. At each step of the induction, we assume that the desired property holds for all subderivations (i.e., that if $\texttt{s} : \mathsf{S}$ and $\texttt{s} \to \texttt{s}'$, then $\texttt{s}' : \mathsf{S}$ whenever $\texttt{s} : \mathsf{S}$ is proved by a subderivation of the present one) then and proceed by case analysis on the final rule in the derivation.

Case T-ZERO:
$\texttt{e} = 0$ and $\texttt{T} = \mathsf{Nat}$
If the last rule in the derivation is T-ZERO, then we know from the form of this rule that e must be a value and T must be : Nat. But if e is a value, then it cannot be the case that $\texttt{e} \to \texttt{e}'$ for any e', and the requirements of the theorem are vacuously satisfied.

Case T-SUCC:
$\texttt{e} = \texttt{succ e1}$, $\texttt{T} = \mathsf{Nat}$ and $\texttt{e1} : \mathsf{Nat}$
By inspecting the evaluation rules, we see that there is just one rule, E-SUCC, that can be used to derive $\texttt{e} \to \texttt{e}'$. The form of this rule tells us that $\texttt{e1} \to \texttt{e1}'$. Since we also know $\texttt{e1} : \mathsf{Nat}$, we can apply the induction hypothesis to obtain $\texttt{e1}' : \mathsf{Nat}$, from which we obtain $\texttt{succ}(\texttt{e1}') : \mathsf{Nat}$, i.e., $\texttt{e}' : \texttt{T}$, by applying rule T-SUCC.

Case T-SUM:
$\texttt{e} = \texttt{e1} + \texttt{e2}$, $\texttt{T} = \mathsf{Nat}$, $\texttt{e1} : \mathsf{Nat}$ and $\texttt{e2} : \mathsf{Nat}$
By inspecting the evaluation rules, we see that there are four rules, E-SUMZERO, E-SUMNV, E-SUMSUCC and E-SUM, that can be used to derive $\texttt{e} \to \texttt{e}'$.

1. Subcase E-SUMZERO:
   $\texttt{e}' = 0 + \texttt{e2}'$, $\texttt{e1} = 0$ and $\texttt{e2} = \texttt{e2}'$
   We know from the form of this rule that $\texttt{e2} \to \texttt{e2}'$ must be the only way and T must be : Nat. We can apply the induction hypothesis to this subderivation, obtaining $\texttt{e2}' : \mathsf{Nat}$. Combining this with the fact that $0 : \mathsf{Nat}$, we can apply rule T-SUM to conclude that $0 + \texttt{e2}' : \mathsf{Nat}$, that is $\texttt{e}' : \mathsf{Nat}$.

2. Subcase E-SUMNV:
   $\texttt{e}' = 0 + \texttt{e2}$ and $\texttt{e1} = 0$
   We know from the form of this rule that $e2$ must be a numeric value and T must be : Nat. But if $e2$ is a value, then it cannot be the case that $\texttt{e2} \to \texttt{e2}'$

for any $e2'$, and the requirements of the theorem are vacuously satisfied.

3. Subcase E-SUMSUCC:
   $\mathtt{e}' = \mathtt{succ(e1)} + \mathtt{e2}$
   We know from the form of this rule that $\mathtt{T}$ must be : Nat for $e2$. By inspecting the evaluation rules, we see that there is just one rule, E-SUCC, that can be used to derive $\mathtt{succ(e1)} \to \mathtt{succ(e1')}$. The form of this rule tells us that $\mathtt{e1} \to \mathtt{e1}'$. Since we also know $\mathtt{e1} : \mathsf{Nat}$, we can apply the induction hypothesis to obtain $\mathtt{e1}' : \mathsf{Nat}$, from which we obtain $\mathtt{succ(e1')} : \mathsf{Nat}$, by applying rule T-SUCC.

4. Subcase E-SUM:
   $\mathtt{e}' = \mathtt{e1}' + \mathtt{e2}$
   From the assumption of the T-SUM case, we have a subderivation of the original typing derivation whose conclusion is $\mathtt{e1} : \mathsf{Nat}$. We know that $\mathtt{e1} \to \mathtt{e1}'$, so we can apply the induction hypothesis to this subderivation, obtaining $\mathtt{e1}' : \mathsf{Nat}$. We can apply rule T-SUM to conclude that $\mathtt{e1}' + \mathtt{e2} : \mathsf{Nat}$, that is $\mathtt{e}' : \mathsf{Nat}$.

Case T-EMPTY:
$\mathtt{e} = \varepsilon$ and $\mathtt{T} = \mathsf{String}$
If the last rule in the derivation is T-EMPTY, then we know from the form of this rule that e must be a value and $\mathtt{T}$ must be : String. But if e is a value, then it cannot be the case that $\mathtt{e} \to \mathtt{e}'$ for any e', and the requirements of the theorem are vacuously satisfied.

Case T-ASTRING:
$\mathtt{e} = \mathtt{Ae1}$, $\mathtt{T} = \mathsf{String}$ and $\mathtt{e1} : \mathsf{String}$
By inspecting the evaluation rules, we see that there is just one rule, E-ASTRING, that can be used to derive $\mathtt{e} \to \mathtt{e}'$. The form of this rule tells us that $\mathtt{A}e\mathtt{1} \to \mathtt{e1}$. Since we also know $\mathtt{e1} : \mathsf{String}$, we can apply the rule T-ASTRING.

Case T-BSTRING:
$\mathtt{e} = \mathtt{Be1}$, $\mathtt{T} = \mathsf{String}$ and $\mathtt{e1} : \mathsf{String}$
By inspecting the evaluation rules, we see that there is just one rule, E-ASTRING, that can be used to derive $\mathtt{e} \to \mathtt{e}'$. The form of this rule tells us that $\mathtt{B}e\mathtt{1} \to \mathtt{e1}$. Since we also know $\mathtt{e1} : \mathsf{String}$, we can apply the rule T-ASTRING.

Case T-LEN:
$\mathtt{e} = \mathtt{len}\,\mathtt{e1}$, $\mathtt{T} = \mathsf{Nat}$ and $\mathtt{e1} : \mathsf{String}$
By inspecting the evaluation rules, we see that there is just one rule, E-LEN, that can be used to derive $\mathtt{e} \to \mathtt{e}'$. The form of this rule tells us that $\mathtt{e1} \to \mathtt{e1}'$. Since we also know $\mathtt{e1} : \mathsf{Nat}$, we can apply the induction hypothesis to obtain $\mathtt{e1}' : \mathsf{String}$, from which we obtain $\mathtt{succ(len(e1'))} : \mathsf{Nat}$, i.e., $\mathtt{e}' : \mathtt{T}$, by applying rule T-LEN.

Case T-LENEMPTY:
$\mathtt{e} = \mathtt{len}\,\varepsilon$ and $\mathtt{T} = \mathsf{Nat}$
If the last rule in the derivation is T-LENEMPTY, then we know from the form of this rule that that can be used to derive $\mathtt{e} \to \mathtt{e}'$, ε is value and $\mathtt{T}$ must be : Nat. Since we also know $\varepsilon : \mathsf{Nat}$, we can apply the induction hypothesis

to $\texttt{l}en\ \varepsilon$: Nat, from which we obtain 0 : Nat, i.e., e' : T, by applying rule T-LENEMPTY.

4 Implementation (30%)

A. Write a data declaration to describe terms. Write Haskell values for the five terms at the left of the arrow in Section 3.**C**.

B. Write predicate val and function eval1. Use these functions to define a function eval that computes the normal form of a term. Predicates must be complete. Functions may yield exceptions when in presence of non-typable terms.

B. Write function typeof that computes the type of a given term if this exists, and raises an exception otherwise.

Due date: October 24, 2022, 23:59