Fixed point combinators and recursive types

Mid-term exam, MPRI 2-4

November 29, 2023 — Duration: 2h45

Answers are judged by their correctness, clarity, conciseness, and accuracy.

Although the questions are in English, it is permitted to answer in French or English. It is recommended to answer in French if this is your mother tongue.

1 Untyped λ-calculus

Throughout this exam, we are interested in pure λ-terms.
The syntax of terms is $t ::= x \mid \lambda x.t \mid t\ t$.
We write $C(t)$ as an abbreviation for the term $\lambda x.t\ (x\ x)$.
We write $Y(t)$ as an abbreviation for the term $C(t)\ C(t)$.
We write *fix* as an abbreviation for the term $\lambda f.Y(f)$.
The term *fix* is known in the literature as Curry's fixed point combinator.
We recall that the call-by-name reduction relation $\longrightarrow_{cbn}$ is defined as follows:

$$\textsc{CbnBeta} \quad (\lambda x.t_1)\ t_2 \longrightarrow_{cbn} [t_2/x]t_1 \qquad\qquad \textsc{CbnAppLeft} \quad \frac{t_1 \longrightarrow_{cbn} t_1'}{t_1\ t_2 \longrightarrow_{cbn} t_1'\ t_2}$$

We write $\longrightarrow^+_{cbn}$ for the transitive closure of the relation $\longrightarrow_{cbn}$.

Question 1 *Prove* $Y(t) \longrightarrow_{cbn} t\ (Y(t))$. Answer □

We write $\mu f.\lambda x.t$ as an abbreviation for $Y(\lambda f.\lambda x.t)$.

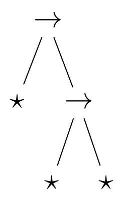
Question 2 *Prove* $\mu f.\lambda x.t \longrightarrow^+_{cbn} [\mu f.\lambda x.t/f](\lambda x.t)$. Answer □

We write *loop* as an abbreviation for $\mu f.\lambda x.f\ x$, that is, $\mu f.\lambda x.(f\ x)$.

Question 3 *Prove* *loop* $t \longrightarrow^+_{cbn}$ *loop* t. *What can be said about the term* (*loop* t)*?* Answer □

2 Simply-typed λ-calculus

In this section, we consider the simply-typed λ-calculus with one base type $\star$.
The syntax of types is $T ::= \star \mid T \to T$. This is an inductive definition.
The letters A, B, T, U, V range over types.
A type is a *finite tree* where every node is either a leaf node $\star$, carrying the label $\star$ and zero subtree, or a binary node $U \to V$, carrying the label $\to$ and two subtrees U and V. For instance, the type $\star \to \star \to \star$ is the following tree:



We recall that the typing judgement $\Gamma \vdash t : T$ is inductively defined by the following three typing rules:

$$\textsc{Var}\quad \Gamma \vdash x : \Gamma(x) \qquad \textsc{Abs}\quad \frac{\Gamma; x : U \vdash t : V}{\Gamma \vdash \lambda x.t : U \to V} \qquad \textsc{App}\quad \frac{\Gamma \vdash t_1 : U \to V \qquad \Gamma \vdash t_2 : U}{\Gamma \vdash t_1\, t_2 : V}$$

There is no introduction rule or elimination rule for the base type $\star$.

Question 4 *By exploiting a property of the simply-typed λ-calculus that has been established in class, explain why the term $(loop\ t)$ cannot be well-typed. A very brief argument is expected.* Answer □

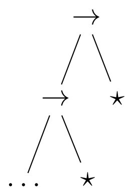
Question 5 *For an arbitrary environment Γ and an arbitrary type B, prove that the typing judgement $\Gamma \vdash x\ x : B$* cannot *hold. A direct argument is expected; this question does not depend on the previous question.* Answer □

3 Simply-typed λ-calculus with recursive types

In this section, we move to the simply-typed λ-calculus with recursive types.

The syntax of types is still $T ::= \star \mid T \to T$, but this time, this definition is understood as a *co-inductive* definition. In other words, a type is now a *possibly infinite tree* where every node is either a leaf node $\star$, carrying the label $\star$ and zero subtree, or a binary node $U \to V$, carrying the label $\to$ and two subtrees U and V.

Some types are infinitely deep: that is, they have one or more infinite branches. For instance, there exists a type A such that the equation $A = A \to \star$ holds. This type is a tree whose leftmost branch is infinite. It can be drawn as follows:



More generally, we remark that every contractive equation $X = F[X]$ admits a unique solution. Let us spell out what this means. Let a *type context* F be defined by the grammar $F ::= [] \mid F \to U \mid T \to F$. Let us say that the equation $X = F[X]$ is *contractive* if F is not the empty context $[]$. Then, there exists a unique type A such that $A = F[A]$ holds.

The typing judgement is still written $\Gamma \vdash t : T$ and is still inductively defined by the same three rules as in Section 2. Now, though, the letters A, B, T, U, V range over possibly infinite types.

Question 6 *Under a certain hypothesis about the types A and B, prove that the typing judgement $x : A \vdash x\ x : B$ holds.* Answer □

Question 7 *Under the same hypothesis about the types A and B, prove the typing judgement $f : B \to B \vdash C(f) : A \to B$.* Answer □

Question 8 *Let B be an arbitrary type. From the previous question, deduce that the typing judgement $\vdash \mathit{fix} : (B \to B) \to B$ holds.* Answer □

4 Simply-typed λ-calculus with restricted recursive types

In this section, we move to a simply-typed λ-calculus that is extended with a new type constructor $\bullet$, pronounced "later", and whose recursive types are *restricted.* Only certain recursive types, the "well-formed" types, are permitted. This type system is due to Nakano (2000).

The syntax of *pre-types* is now $T ::= \star \mid \bullet T \mid T \to T$. As in the previous section, this definition is understood as a *co-inductive* definition. So, a pre-type is a *possibly infinite tree* where every node is:

1. a leaf node $\star$, carrying the label $\star$ and zero subtree, or
2. a unary node $\bullet U$, carrying the label $\bullet$ and one subtree U, or
3. a binary node $U \to V$, carrying the label $\to$ and two subtrees U and V.

A pre-type is *well-formed* if, starting from any node, every branch eventually reaches a node labeled with $\star$ or $\bullet$.

A *type* is a well-formed pre-type.

Question 9 *Give an example of a pre-type (an infinite tree) that is not well-formed. You may give a mathematical definition of this infinite tree as the unique solution of a contractive equation, or you may just draw it.* Answer □

The syntax of type contexts is extended: $F ::= [] \mid \bullet F \mid F \to U \mid T \to F$. It is still true that every contractive equation $X = F[X]$ admits a unique solution: that is, there exists a unique pre-type A such that $A = F[A]$ holds. However, A is not necessarily well-formed; one must check that it is.

4.1 Type-checking the fixed point combinator

Question 10 *Let B be an arbitrary type. Prove that there exists a (unique) type A such that the equation $A = \bullet(A \to B)$ holds.* Answer □

From this point on, we restrict our attention to types, as opposed to pre-types. The letters A, B, T, U, V range over types. An ill-formed pre-type never appears anywhere.

From here on, the typing judgement, still written $\Gamma \vdash t : T$, is inductively defined by the following *four* typing rules. They are the three earlier rules, plus a new rule, Sub:

$$\textsc{Var}\quad \Gamma \vdash x : \Gamma(x) \qquad \textsc{Abs}\quad \frac{\Gamma; x : U \vdash t : V}{\Gamma \vdash \lambda x.t : U \to V} \qquad \textsc{App}\quad \frac{\Gamma \vdash t_1 : U \to V \quad \Gamma \vdash t_2 : U}{\Gamma \vdash t_1\, t_2 : V} \qquad \textsc{Sub}\quad \frac{\Gamma \vdash t : T \qquad T \leq T'}{\Gamma \vdash t : T'}$$

The definition of the subtyping relation $T \leq T'$ is omitted, because it is technical. We admit that this relation is reflexive, transitive, satisfies the following four properties:

$$\textsc{LaterIntro}\quad A \leq \bullet A \qquad \textsc{LaterArrow}\quad \bullet(A \to B) \approx \bullet A \to \bullet B \qquad \textsc{LaterVariance}\quad \frac{A \leq B}{\bullet A \leq \bullet B} \qquad \textsc{ArrowVariance}\quad \frac{A' \leq A \qquad B \leq B'}{A \to B \leq A' \to B'}$$

In LaterArrow, we write $A \approx B$ to mean that both $A \leq B$ and $B \leq A$ hold.

A subtyping judgement $T \leq T'$ intuitively means that the type T is more precise, or "better", than the type T'. Thus, a term of type T can also be viewed as a term of type T': this is expressed by the typing rule Sub. Thus, for instance, LaterIntro and Sub together imply that a term of type T also has type $\bullet T$. The converse is false.

The next three questions revisit Questions 6, 7, and 8 in the setting of Nakano's type system.

Question 11 *Assuming that the equation* $A = \bullet(A \to B)$ *holds, prove that the typing judgement* $x : A \vdash x\ x : \bullet B$ *holds.* Answer □

Question 12 *Still assuming that the equation* $A = \bullet(A \to B)$ *holds, establish the typing judgement* $f : \bullet B \to B \vdash C(f) : A \to B$. Answer □

Question 13 *Let B be an arbitrary type. Prove* $\vdash \mathit{fix} : (\bullet B \to B) \to B$. Answer □

Question 14 *Is the term* $(\mathit{fix}\ (\lambda x.x))$ *well-typed or ill-typed? Explain why. A brief answer is expected.* Answer □

4.2 Proving subject reduction

The following questions are independent of the previous questions.

The next question concerns a *subtyping inversion* lemma for functions. Such a lemma is intended to extract information out of a subtyping relation between two function types, $T \to U \leq T' \to U'$.

Question 15 *By giving a counter-example, prove that this candidate statement is* false*: "$T \to U \leq T' \to U'$ implies $T' \leq T$ and $U \leq U'$". Then, suggest a plausible repaired statement of the form "$T \to U \leq T' \to U'$ implies ... ". No proof of this repaired statement is requested.* Answer □

The next question aims to prove subject reduction. For simplicity, we focus on the special case of a β-reduction at the root, that is, under an empty context. We write $\longrightarrow_\beta$ for the reduction relation defined by the single axiom $(\lambda x.t_1)\ t_2 \longrightarrow_\beta [t_2/x]t_1$.

Question 16 *Prove that* $\Gamma \vdash t : T$ *and* $t \longrightarrow_\beta t'$ *imply* $\Gamma \vdash t' : T$. *If you find that one or more auxiliary lemmas are needed, give the statement of each necessary lemma, without proving it.* Answer □

4.3 Defining a logical relation

The next question is independent of the previous questions. It concerns the definition of a step-indexed unary logical relation, also known as a "semantic typing judgement".

We fix a certain set of terms K.

We wish to define a judgement $\vDash_n t : T$ that satisfies the following four properties:

1. $\vDash_0 t : T$ always holds.
2. $\vDash_{n+1} t : \star$ holds if and only if $t \in K$ holds.
3. $\vDash_{n+1} t : \bullet T$ holds if and only if $\vDash_n t : T$ holds.

4. $\vDash_{n+1} t : U \rightarrow V$ holds if and only if
   for every $i \leq n+1$ and for every term u,
   $\vDash_i u : U$ implies $\vDash_i t\, u : V$.

However, it is not obvious whether and why the above four properties form an acceptable *definition* of the 3-place relation $\vDash_n t : T$. The following two questions concern this issue.

Question 17 *Define a strict order* $(n, t, T) > (n', t', T')$ *such that, in each of the above four properties, the semantic judgement on the left-hand side is greater than the semantic judgement(s) that appear on the right-hand side. Show that this order is well-founded, that is, it does not admit an infinite descending chain.* Answer □

Let us remark that when we defined the syntactic typing judgement $\Gamma \vdash t : T$, we did not justify this definition by exhibiting a well-founded order on (Γ, t, T). In fact, such a well-founded order does not exist. Instead, we defined the syntactic typing judgement $\Gamma \vdash t : T$ as the *smallest* relation that satisfies all of the typing rules. This is known as an *inductive definition*. This kind of definition does not require a well-founded order.

Question 18 *To justify that properties 1–4 form a meaningful definition, is it necessary to exhibit a well-founded order, as done in the previous question? Could one instead view properties 1–4 as an* inductive definition *of the semantic typing judgement? A brief answer is expected.* Answer □

That was the last question!

The following is cultural background.

One defines the plain (non-indexed) semantic typing judgement $\vDash t : T$ as a shorthand for $\forall n.\ \vDash_n t : T$.

Provided that the set K is suitably chosen, one can prove that syntactic typing implies semantic typing: $\Gamma \vdash t : T$ implies $\vDash t : T$. (The term t is not necessarily closed.)

Furthermore, provided the set K is suitably chosen, one can prove that semantic typing implies the existence of a weak head normal form: $\vDash t : T$ implies that t can be reduced to a weak head normal form, where a weak head normal form is either a λ-abstraction or a term of the form $x\, t_1 \ldots\, t_n$.

Thus, even though the fixed point combinator *fix* is well-typed, this type system rules out divergent computations, such as *loop*() (Question 3) or *fix* $(\lambda x.x)$. The type assigned to *fix*, which is $(\bullet B \rightarrow B) \rightarrow B$, requires the recursive computation to be *productive*: intuitively, the result of the recursive call can be demanded only "in the next time step", not "now".

When a well-typed term t is reduced to a weak head normal form $x\, t_1 \ldots\, t_n$, the subterms $t_1, \ldots, t_n$ are not necessarily normalized in any way. However, by subject reduction, they must be well-typed, so each of them can itself be reduced to a weak head normal form, and so on. In i time steps, the term t can be normalized down to depth i.

A recursive type of productive streams can be defined: it is characterized by the recursive equation $\mathit{Stream}\ A = 1 + A \times \bullet(\mathit{Stream}\ A)$. If a term t has type $\mathit{Stream}\ A$, then in i time steps, the i-th element of this stream can be computed.

In logic, the statement $(\bullet B \rightarrow B) \rightarrow B$ is known as Löb's induction principle.

5 Solutions

Question 1

$$\begin{array}{rcll}
Y(t) & = & C(t)\;C(t) & \text{by definition of } Y(t) \\
 & = & (\lambda x.t\;(x\;x))\;C(t) & \text{by definition of } C(t) \\
 & \longrightarrow_{cbn} & t\;(C(t)\;C(t)) & \text{by } \textsc{CbnBeta} \\
 & = & t\;(Y(t)) & \text{by definition of } Y(t)
\end{array}$$

Question 2

$$\begin{array}{rcll}
\mu f.\lambda x.t & = & Y(\lambda f.\lambda x.t) & \text{by definition of } \mu f.\lambda x.t \\
 & \longrightarrow_{cbn} & (\lambda f.\lambda x.t)\;(Y(\lambda f.\lambda x.t)) & \text{by Question 1} \\
 & = & (\lambda f.\lambda x.t)\;(\mu f.\lambda x.t) & \text{by definition of } \mu f.\lambda x.t \\
 & \longrightarrow_{cbn} & [\mu f.\lambda x.t/f](\lambda x.t) & \text{by } \textsc{CbnBeta}
\end{array}$$

Question 3

First, we remark that:

$$\begin{array}{rcll}
\mathit{loop} & = & \mu f.\lambda x.f\;x & \text{by definition of } \mathit{loop} \\
 & \longrightarrow^{+}_{cbn} & [\mathit{loop}/f](\lambda x.f\;x) & \text{by Question 2 and by definition of } \mathit{loop} \\
 & = & \lambda x.\mathit{loop}\;x & \text{by definition of substitution}
\end{array}$$

Now, call-by-name reduction allows reduction in the left-hand side of an application; this is expressed by CbnAppLeft. Thus, we have:

$$\begin{array}{rcll}
\mathit{loop}\;t & \longrightarrow^{+}_{cbn} & (\lambda x.\mathit{loop}\;x)\;t & \text{by } \textsc{CbnAppLeft} \text{ and the previous remark} \\
 & \longrightarrow_{cbn} & \mathit{loop}\;t & \text{by } \textsc{CbnBeta}
\end{array}$$

Thus, the term $\mathit{loop}\;t$ admits an infinite reduction sequence: it diverges.

Question 4

By Question 3, the term $(\mathit{loop}\;t)$ diverges under call-by-name reduction. However, we have established in class that the simply-typed λ-calculus has strong normalization: that is, a well-typed term cannot have an infinite reduction sequence, even when reduction under arbitrary contexts is permitted. Therefore, $(\mathit{loop}\;t)$ cannot be well-typed.

Question 5

Assume that $\Gamma \vdash x\;x : B$ holds. Then, by inversion of the typing rule App, for some type A, we must have $\Gamma \vdash x : A \rightarrow B$ and $\Gamma \vdash x : A$. By inversion of the typing rule Var, we must then have $\Gamma(x) = A \rightarrow B$ and $\Gamma(x) = A$. From these equations, we deduce, by transitivity of equality, $A = A \rightarrow B$. This, however, is impossible: because types are inductively defined, they are finite trees; the type A cannot be a subtree of itself.

Question 6

We have discovered at Question 5 that, for this judgement to hold, the equation $A = A \to B$ is necessary. Conversely, let us assume that $A = A \to B$ holds. Then, by the typing rule Var, we have $x : A \vdash x : A \to B$ and $x : A \vdash x : A$. So, by the typing rule App, we obtain $x : A \vdash x\ x : B$.

Question 7

The goal is:

$$f : B \to B \vdash C(f) : A \to B$$

By definition of $C(f)$, this is:

$$f : B \to B \vdash \lambda x.f\ (x\ x) : A \to B$$

By the typing rule Abs, this goal can be reduced to:

$$f : B \to B; x : A \vdash f\ (x\ x) : B$$

By the typing rule App, this goal follows from the two subgoals:

$$f : B \to B; x : A \vdash f : B \to B \qquad \text{and} \qquad f : B \to B; x : A \vdash x\ x : B$$

The first subgoal follows from the typing rule Var. The second one follows from Question 6 and from a weakening lemma.

Question 8

By definition of *fix*, the goal is:

$$\vdash \lambda f.Y(f) : (B \to B) \to B$$

By the typing rule Abs, this goal boils down to:

$$f : B \to B \vdash Y(f) : B$$

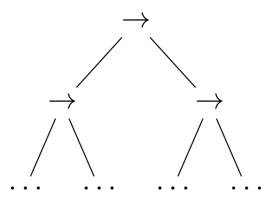
By definition of $Y(f)$, this goal can also be written:

$$f : B \to B \vdash C(f)\ C(f) : B$$

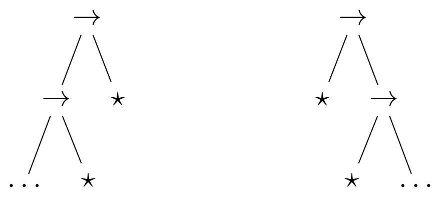
Now, by Question 7, we have $f : B \to B \vdash C(f) : A \to B$ (**1**), where A is the (unique) type such that $A = A \to B$ holds. Furthermore, because this equation holds, the typing judgement (1) can also be written under the form $f : B \to B \vdash C(f) : A$ (**2**). By applying the typing rule App to the judgements (1) and (2), we obtain the goal.

Question 9

Let A stand for the (unique) tree such that $A = A \to A$ holds. It is an infinite tree where every node is a binary node labeled with $\to$. Every branch in this tree is infinite, and no branch ever reaches a node labeled with $\star$ or $\bullet$. Therefore, this type is not well-formed.



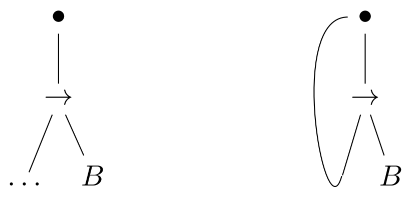
As two other examples, the types characterized by the equations $A = A \to \star$ and $A = \star \to A$ are not well-formed either. Some of their branches are finite and end at a node labeled $\star$, but each of them has an infinite branch that never reaches a node labeled with $\star$ or $\bullet$.



Question 10

Let F stand for the type context $\bullet([] \to B)$. This type context is nonempty, so the equation $X = F[X]$ is contractive. Therefore, there exists a (unique) pre-type A such that $A = F[A]$ holds. This equation can also be written $A = \bullet(A \to B)$.

The pre-type A can be drawn in two ways, as follows. In the drawing on the left-hand side, the ellipsis ... represents a copy of A itself. In the drawing on the right-hand side, this has been represented by a backward edge towards the root instead of an ellipsis.



There remains to argue that A is not just a pre-type but also a type. That is, we must check that A is well-formed.

To prove that A is well-formed, we must prove that, by starting at an arbitrary node inside A and by descending down an arbitrary branch, one must eventually reach a node labeled with $\star$ or $\bullet$. In the above drawing, at right, it is clear that this is true: indeed, starting anywhere, either one follows the backward edge and reaches the root node, which carries the label $\bullet$, or one descends into B, where, because B itself is well-formed, one must eventually reach a node labeled with $\star$ or $\bullet$.

Question 11

By the typing rule Var, we have $x : A \vdash x : A$ (**3**). According to the equation $A = \bullet(A \to B)$, the judgement (3) can also be written $x : A \vdash x : \bullet(A \to B)$, which by Sub and LaterArrow implies $x : A \vdash x : \bullet A \to \bullet B$ (**4**). Besides, by Sub and LaterIntro, the judgement (3) also implies $x : A \vdash x : \bullet A$ (**5**). Applying the typing rule App to the judgements (4) and (5) yields $x : A \vdash x\ x : \bullet B$.

Question 12

The goal is:

$$f : \bullet B \to B \vdash C(f) : A \to B$$

By definition of $C(f)$, this goal can be written:

$$f : \bullet B \to B \vdash \lambda x.f\ (x\ x) : A \to B$$

By the typing rule Abs, the goal can be reduced to:

$$f : \bullet B \to B; x : A \vdash f\ (x\ x) : B$$

By the typing rule App, this goal follows from the two subgoals:

$$f : \bullet B \to B; x : A \vdash f : \bullet B \to B \qquad \text{and} \qquad f : \bullet B \to B; x : A \vdash x\ x : \bullet B$$

The first subgoal follows from the typing rule Var. The second one follows from Question 11, up to weakening.

Question 13

The goal is:

$$\vdash \mathit{fix} : (\bullet B \to B) \to B$$

By definition of *fix*, this goal can be written:

$$\vdash \lambda f.Y(f) : (\bullet B \to B) \to B$$

By the typing rule Abs, this goal boils down to:

$$f : \bullet B \to B \vdash Y(f) : B$$

By definition of $Y(f)$, this goal can also be written:

$$f : \bullet B \to B \vdash C(f)\ C(f) : B$$

By Question 10, there exists a (unique) type A such that the equation $A = \bullet(A \to B)$ holds. By Question 12, we have $f : \bullet B \to B \vdash C(f) : A \to B$ (**6**). By the subtyping rule LaterIntro, we have $A \to B \leq \bullet(A \to B)$, that is, $A \to B \leq A$. Thus, by Sub, the judgement (6) implies $f : \bullet B \to B \vdash C(f) : A$ (**7**). By applying the typing rule App to the judgements (6) and (7), we obtain the goal.

Question 14

The term $(\mathit{fix}\ (\lambda x.x))$ is ill-typed. Because *fix* has type $(\bullet B \to B) \to B$, type-checking $(\mathit{fix}\ (\lambda x.x))$ would require the identity function to have type $\bullet B \to B$. This in turn would require the subtyping relation $\bullet B \leq B$. However, this relation does not hold.

One may wonder whether it is possible to give *fix* a stronger type, such as $(B \to B) \to B$, in which case $\mathit{fix}\ (\lambda x.x)$ would be well-typed. This is in fact not possible. The term $\mathit{fix}\ (\lambda x.x)$ diverges; yet, using the logical relation defined in Section 4.3, one can prove that every well-typed term admits a head normal form (therefore does not diverge).

Question 15

In the ordinary simply-typed λ-calculus with subtyping, without a "later" type constructor, the subtyping relation $T \to U \leq T' \to U'$ implies $T' \leq T$ and $U \leq U'$. However, here, this implication is not true: as a counter-example, $T \to U \leq \bullet T \to \bullet U$ is true (it follows from the rules LaterIntro and LaterArrow), yet $\bullet T \leq T$ does not hold. This is a counter-example.

More generally, we can give a family of counter-examples: by iterating LaterIntro and LaterArrow, we have $T \to U \leq \bullet^n (T \to U) \leq \bullet^n T \to \bullet^n U$ for every natural number n.

Therefore, a plausible statement of a subtyping inversion lemma is the following (written in the form of a deduction rule):

$$\frac{T \to U \leq T' \to U'}{\exists n. \quad T' \leq \bullet^n T \wedge \bullet^n U \leq U'}$$

This lemma is indeed true, but the proof cannot be given here, since the definition of the subtyping relation has not been provided. (We have only provided six properties that the subtyping relation satisfies.)

Question 16

By definition of the relation $\longrightarrow_\beta$, the goal can be formulated as follows: assuming $\Gamma \vdash (\lambda x.t_1)\ t_2 : U'$ (**8**), prove $\Gamma \vdash [t_2/x]t_1 : U'$. (We have renamed T to U'.) We prove this implication by induction over the derivation of the typing judgement (8). This judgement can be obtained only via the rules Sub or App, so only these two cases must be considered.

Case Sub. This case is trivial; it is just a matter of applying the induction hypothesis and applying Sub again.

Case App. For some type T', the premises of App must be $\Gamma \vdash \lambda x.t_1 : T' \to U'$ (**9**) and $\Gamma \vdash t_2 : T'$ (**10**). The judgement (9) must follow from Abs and zero, one, or more instances of Sub. Because subtyping is reflexive and transitive, we may in fact assume that exactly one instance of Sub has been used. Thus, for some types T and U, we must have $T \to U \leq T' \to U'$ (**11**) and $\Gamma; x : T \vdash t_1 : U$ (**12**).

By applying the subtyping inversion lemma (Question 15) to the subtyping relation (11), we find that, for some natural number n, the subtyping judgements $T' \leq \bullet^n T$ (**13**) and $\bullet^n U \leq U'$ (**14**) hold.

Applying Sub to the judgements (10) and (13) yields $\Gamma \vdash t_2 : \bullet^n T$ (**15**).

Then, from $\Gamma; x : T \vdash t_1 : U$ (12), we need to get $\Gamma; x : \bullet^n T \vdash t_1 : \bullet^n U$ (**16**). This requires an auxiliary lemma, which can be stated as follows: "$\Gamma_1; \Gamma_2 \vdash t : T$ implies $\Gamma_1; \bullet\Gamma_2 \vdash t : \bullet T$".

Applying Sub to the judgements (16) and (14) yields $\Gamma; x : \bullet^n T \vdash t_1 : U'$ (**17**).

Applying a standard term-for-variable substitution lemma (another auxiliary lemma) to the judgements (17) and (15) yields $\Gamma \vdash [t_2/x]t_1 : U'$. This was our goal.

Question 17

Let us first remark that the desired order cannot be based on the natural number n alone, because some of the recursive calls do not cause a decrease in n. (This is the case of the two calls in property 4, where i can be $n+1$.)

Let us also remark that the desired order also cannot be based on a naive notion of the "size" of the type T. A type is not an inductive object: it is a potentially infinite tree. The types U and V are not "smaller" in any sense than the type $U \to V$.

Let us finally remark that the desired order cannot be based on the term t, since the size of this term does not decrease in property 3, and actually increases in property 4.

A suitable order can be defined based on the pair (n, T). (The middle component, the term t, can be ignored.) In fact, regarding the definition of this order, there is no choice. The order that must be considered is generated by the following rules:

$$(n+1, \bullet T) > (n, T) \qquad \frac{n+1 \geq i}{(n+1, U \to V) > (i, U)} \qquad \frac{n+1 \geq i}{(n+1, U \to V) > (i, V)}$$

These rules are obtained mechanically by inspecting the recursive calls in properties 3 and 4. The first rule is necessary for the recursive call in property 3 to be decreasing; the last two rules are necessary for the recursive calls in property 4 to be decreasing.

Now, there remains to check that this order is well-founded, that is, it does not admit an infinite descending chain $(n, T) > \ldots > \ldots$ Assume, by way of contradiction, that there exists such an infinite chain. Clearly, this chain cannot infinitely often go through a "later" constructor, because the integer index decreases every time a "later" constructor is crossed. Thus, this chain must eventually stop exploiting the first rule and use only the last two rules. This implies that there exists in the type T an infinite branch where every node is an "arrow" node. But this is impossible: this contradicts the fact that the type T must be well-formed. Indeed, well-formedness guarantees that every branch eventually reaches a node labeled with $\star$ or $\bullet$.

Thus, the well-formedness of types is necessary for this order to be well-founded, and is therefore necessary for properties 1–4 to form a correct recursive definition.

Question 18

Viewing properties 1–4 as an inductive definition would mean that the judgement $\vDash_n t : T$ is defined as the *smallest* relation that satisfies properties 1–4. However, it is not clear whether the set S of "the relations that satisfy properties 1–4" is nonempty and whether it is closed under intersections. So, it is not clear whether it has a smallest inhabitant.

The root of the problem is that the semantic typing judgement occurs in a *negative* position in property 4, "$\vDash_i u : U$ implies ...". In Coq, an attempt to define the semantic typing judgement using the `Inductive` keyword would be rejected because of this negative occurrence.

If there wasn't such a negative occurrence, then the set S would be nonempty (it would be inhabited by the relation that is everywhere true) and closed under intersections, so it would indeed have a smallest inhabitant.