

Compilation d'un langage avec sous-typage

Compiling a language with subtyping

Examen du cours MPRI 2-4 / Exam for MPRI 2-4 course

2011/03/08 — Durée / duration: 2h30

Le but de ce problème est d'étudier une technique de compilation des langages avec sous-typage, basée sur l'insertion de conversions du sous-type vers le super-type. De ce point de vue, on peut distinguer deux sortes de sous-typage :

- Sans changement de représentation : par exemple, supposons un type `pos` des entiers positifs, qui est sous-type du type `int` des entiers relatifs. Vraisemblablement, un entier positif va avoir la même représentation en machine qu'il soit vu avec le type `pos` ou avec le type `int`. Donc, il n'est pas nécessaire de produire du code de conversion pour une étape de subsumption faisant passer de $a : \text{pos}$ à $a : \text{int}$.
- Avec changement de représentation : c'est le cas par exemple si l'on considère le type `int` comme sous-type du type `float` des nombres flottants. L'entier 3 n'a pas la même représentation dans la machine que le flottant 3.0. Il faut donc, à l'exécution, effectuer une conversion entier vers flottant lorsqu'une expression $a : \text{int}$ est vue, par subsumption, comme $a : \text{float}$.

Dans ce problème, nous allons voir comment le compilateur peut insérer automatiquement les bonnes conversions entier-flottant en s'appuyant sur les informations de typage du programme.

Le langage source est le λ -calcul avec constantes et types simples. Voici sa syntaxe :

Expressions: $a ::= x \mid c \mid \lambda x : \tau. a \mid a_1 a_2$

Constants: $c ::= 0 \mid 1 \mid \dots \mid 3.1415 \mid 2.718 \mid \dots \mid \text{floatofint}$

Types: $\tau ::= \text{int} \mid \text{float} \mid \tau_1 \rightarrow \tau_2$

Les constantes contiennent les nombres entiers, les nombres flottants, ainsi que l'opérateur `floatofint`. Les types des constantes sont :

The purpose of this exam is to study a compilation technique for languages with subtyping, based on the insertion of coercions from subtype to super-type. In this respect, we can distinguish two kinds of subtyping:

- Without change of representation: for example, assume a type `pos` of positive integers, which is subtype of the type `int` of relative integers. Probably, a positive integer has the same machine representation whether it is viewed with type `pos` or with type `int`. Therefore, it is unnecessary to produce code to coerce a for a subsumption step going from $a : \text{pos}$ to $a : \text{int}$.
- With change of representation: this is the case for instance if we view `int` as a subtype of the type `float` of floating-point numbers. The integer 3 and the floating-point number 3.0 have different machine representations. Therefore, at run-time, we must execute an integer-to-float conversion when an expression $a : \text{int}$ is viewed, by subsumption, as $a : \text{float}$.

In this exam, we shall see how the compiler can automatically insert the correct int-float coercions based on typing information.

The source language is λ -calculus with constants and simple types. Its syntax is as follows:

Constants include integer and float numbers as well as the `floatofint` operator

`0 : int` `3.1415 : float` `floatofint : int → float`

Pour la relation de sous-typage $\tau_1 <: \tau_2$, nous disons que `int` est sous-type de `float` et étendons ce sous-typage aux types de fonctions de la manière standard.

We state that `int` is a subtype of `float` and extend the relation to arrow types as customary.

`int <: float`

`int <: int`

`float <: float`

$$\frac{\sigma' <: \sigma \quad \tau <: \tau'}{\sigma \rightarrow \tau <: \sigma' \rightarrow \tau'}$$

Si $\tau <: \tau'$ et a est un terme de type τ , nous définissons la conversion de a du sous-type τ vers le super-type τ' , notée $\mathcal{C}(\tau, \tau', a)$, de la manière suivante :

If $\tau <: \tau'$ and a is a term of type τ , we define the conversion $\mathcal{C}(\tau, \tau', a)$ for a from the subtype τ to its supertype τ' , as follows:

$$\begin{aligned} \mathcal{C}(\text{int}, \text{float}, a) &= \text{floatofint}(a) \\ \mathcal{C}(\text{int}, \text{int}, a) &= a \\ \mathcal{C}(\text{float}, \text{float}, a) &= a \\ \mathcal{C}(\sigma \rightarrow \tau, \sigma' \rightarrow \tau', a) &= \lambda x : \sigma'. ??? \end{aligned}$$

Enfin, le prédicat de typage et de traduction $\Gamma \vdash a : \tau \Rightarrow b$ est défini par les règles ci-dessous. Ce prédicat est semblable au prédicat de typage habituel $\Gamma \vdash a : \tau$, mais calcule en plus une expression b (la “traduction” de a) qui est l’expression source a dans laquelle on a inséré les conversions `floatofint` explicitant les étapes de sous-typage effectuées durant le typage (voir la règle (sub)).

Finally, we consider the typing/translation judgement $\Gamma \vdash a : \tau \Rightarrow b$, which is derived by the deduction system below. This judgment is similar to the usual typing judgment $\Gamma \vdash a : \tau$, but it further computes an expression b (the “translation” of a) which is obtained from a by inserting the conversions `floatofint` that make explicit the use of the subsumption rule in typing a (see the (sub) rule).

$$\begin{aligned} \Gamma \vdash x : \Gamma(x) \Rightarrow x \quad (\text{var}) & \quad \frac{c : \tau}{\Gamma \vdash c : \tau \Rightarrow c} \quad (\text{const}) \\ \frac{\Gamma \vdash a_1 : \tau' \rightarrow \tau \Rightarrow b_1 \quad \Gamma \vdash a_2 : \tau' \Rightarrow b_2}{\Gamma \vdash a_1 a_2 : \tau \Rightarrow b_1 b_2} \quad (\text{app}) \end{aligned}$$

$$\begin{aligned} \frac{\Gamma; x : \tau' \vdash a : \tau \Rightarrow b}{\Gamma \vdash \lambda x : \tau'. a : \tau' \rightarrow \tau \Rightarrow \lambda x : \tau'. b} \quad (\text{fun}) \\ \frac{\Gamma \vdash a : \tau \Rightarrow b \quad \tau <: \tau'}{\Gamma \vdash a : \tau' \Rightarrow \mathcal{C}(\tau, \tau', b)} \quad (\text{sub}) \end{aligned}$$

Question 1

On se donne une constante `cos` de type `float → float`. Donner une dérivation de typage et de traduction de l’expression `cos 1` où `1` est une constante de type `int`. Quelle est la traduction de cette expression ?

We add a constant `cos` with type `float → float`. Give a typing/translation derivation for the expression `cos 1` where `1` is a constant of type `int`. What is the translation of this expression?

Question 2

Compléter la définition de la conversion entre deux types fonctionnels :

$$\mathcal{C}(\sigma \rightarrow \tau, \sigma' \rightarrow \tau', a) = \lambda x : \sigma'. ???$$

avec $\sigma' <: \sigma$ et $\tau <: \tau'$. On pourra prendre l'exemple d'une fonction $f : \text{float} \rightarrow \text{int}$. Que faire pour pouvoir l'utiliser dans un contexte qui attend une fonction de type $\text{int} \rightarrow \text{float}$?

Fill in the blank in the definition of coercions between two function types:

where $\sigma' <: \sigma$ and $\tau <: \tau'$. For inspiration, you could consider a function $f : \text{float} \rightarrow \text{int}$. What needs to be done to be able to use f in a context expecting a function of type $\text{int} \rightarrow \text{float}$?

Question 3

Dans cette question, on étend le langage source avec des types produits $\tau_1 \times \tau_2$ et trois constantes **pair** (construction d'une paire), π_1 (projection de la première composante d'une paire) et π_2 (projection de la seconde composante). Ces constantes ont les types suivants, où τ_1 et τ_2 sont des types quelconques :

$$\text{pair} : \tau_1 \rightarrow \tau_2 \rightarrow \tau_1 \times \tau_2 \quad \pi_1 : \tau_1 \times \tau_2 \rightarrow \tau_1 \quad \pi_2 : \tau_1 \times \tau_2 \rightarrow \tau_2$$

On étend également la relation de sous-typage avec les règles suivantes (où \times a une priorité plus haute que \rightarrow) :

$$\begin{array}{l} \tau_1 \times \tau_2 <: \tau_1 \\ \tau_1 \times \tau_2 <: \tau_2 \\ \tau_1 \times \tau_2 \rightarrow \tau_3 <: \tau_1 \rightarrow \tau_2 \rightarrow \tau_3 \\ \tau_1 \rightarrow \tau_2 \rightarrow \tau_3 <: \tau_1 \times \tau_2 \rightarrow \tau_3 \\ \tau_1 \rightarrow \tau_2 \times \tau_3 <: (\tau_1 \rightarrow \tau_2) \times (\tau_1 \rightarrow \tau_3) \\ (\tau_1 \rightarrow \tau_2) \times (\tau_1 \rightarrow \tau_3) <: \tau_1 \rightarrow \tau_2 \times \tau_3 \end{array} \quad \frac{\sigma_1 <: \tau_1 \quad \sigma_2 <: \tau_2}{\sigma_1 \times \sigma_2 <: \tau_1 \times \tau_2}$$

Définir la conversion $\mathcal{C}(\tau, \tau', a)$ dans chacun de ces 7 cas de sous-typage $\tau <: \tau'$.

In this question we extend the source language with product types of the form $\tau_1 \times \tau_2$ and three constants **pair** (the constructor of a pair), π_1 (first projection) and π_2 (second projection). These constants have the following types, for all types τ_1 and τ_2 :

We also extend the subtyping relation by the following rules (where the priority of \times is higher than that of \rightarrow):

For each of these 7 cases, define the corresponding conversion $\mathcal{C}(\tau, \tau', a)$.

Question 4

On note $\Gamma \vdash b : \tau$ le système de typage simple, sans sous-typage ni règle de subsomption, défini par les règles suivantes :

$$\Gamma \vdash x : \Gamma(x) \quad \frac{c : \tau}{\Gamma \vdash c : \tau} \quad \frac{\Gamma; x : \tau' \vdash b : \tau}{\Gamma \vdash \lambda x : \tau'. b : \tau' \rightarrow \tau} \quad \frac{\Gamma \vdash b_1 : \tau' \rightarrow \tau \quad \Gamma \vdash b_2 : \tau'}{\Gamma \vdash b_1 b_2 : \tau}$$

Let $\Gamma \vdash b : \tau$ be the simple judgment derived by the following system that does not use either subsomption or subtyping:

Montrer que si $\Gamma \vdash a : \tau \Rightarrow b$, alors $\Gamma \vdash b : \tau$.

Show that $\Gamma \vdash a : \tau \Rightarrow b$ implies $\Gamma \vdash b : \tau$.

Cohérence de la traduction / Coherence of translation

On s'intéresse maintenant au problème de la *cohérence* de la traduction : dans la mesure où il y a plusieurs dérivations de typage-traduction possibles pour une même expression (on peut appliquer la règle (sub) à différents endroits), on peut obtenir plusieurs traductions différentes pour une même expression de départ ; est-ce que toutes ces traductions sont cohérentes, c'est-à-dire calculent le même résultat ?

Question 5

On considère à nouveau l'expression `cos 1`. Donner une autre dérivation de typage et de traduction que celle trouvée à la question 1. Obtenez-vous le même terme traduit ?

Definition

On note \approx la relation de $\beta\eta$ -équivalence entre expressions. Cette équivalence est définie comme suit :

$$\begin{array}{l}
 (\lambda x : \tau. a) b \approx a[x \leftarrow b] \quad (\beta) \\
 \frac{a_1 \approx a_2}{(\lambda x : \tau. a_1) \approx (\lambda x : \tau. a_2)} \quad (\text{equiv-abstr}) \\
 a \approx a \quad (\text{equiv-refl}) \\
 \frac{a_1 \approx a_2}{a_2 \approx a_1} \quad (\text{equiv-sym})
 \end{array}
 \qquad
 \begin{array}{l}
 (\lambda x : \tau. a x) \approx a \quad (x \notin FV(a)) \quad (\eta) \\
 \frac{a_1 \approx a_2 \quad b_1 \approx b_2}{a_1 b_1 \approx a_2 b_2} \quad (\text{equiv-app}) \\
 \frac{a_1 \approx a_2 \quad a_2 \approx a_3}{a_1 \approx a_3} \quad (\text{equiv-trans})
 \end{array}$$

Dans les questions suivantes, nous allons montrer que toutes les traductions possibles pour une même expression à un même type sont $\beta\eta$ -équivalentes

Question 6

Montrer les deux propriétés ci-dessous des conversions :

$$\begin{array}{l}
 \forall \tau, a, \quad \mathcal{C}(\tau, \tau, a) \approx a \\
 \forall \tau_1, \tau_2, \tau_3, a, \quad \tau_1 <: \tau_2 \wedge \tau_2 <: \tau_3 \implies \mathcal{C}(\tau_1, \tau_3, a) \approx \mathcal{C}(\tau_2, \tau_3, \mathcal{C}(\tau_1, \tau_2, a))
 \end{array}$$

We now consider the problem of *coherence* of the translation: since the same expression admits different typing/translation derivations (because of the (sub) rule), then different translations of the same a are possible. We want to check whether all these translations are coherent, that is, they yield the same result.

Consider the expression `cos 1` again. Give it a typing/translating derivation different from the one given in question 1. Do you obtain the same translated term?

Let \approx denote the $\beta\eta$ -equivalence relation over expressions defined as follows:

In what follows we are going to prove that all translations for a given expression a at a given type τ are $\beta\eta$ -equivalent.

Prove the following two properties of the conversions:

Question 7

On considère une dérivation de typage-traduction qui se termine par une règle (sub) suivie d'une règle (fun) :

$$\begin{array}{c} \text{derivation } D \\ \vdots \\ \hline \dots <: \dots \quad (\text{sub}) \\ \dots \\ \hline \Gamma \vdash a : \tau \Rightarrow b \quad (\text{fun}) \end{array}$$

Consider a typing-translation derivation ending by a (sub) rule immediately followed by a (fun) rule:

Montrer que l'on peut permuter les règles (sub) et (fun), obtenant une dérivation de la forme suivante :

Show that (sub) and (fun) can be soundly permuted, yielding a derivation of the following shape:

$$\begin{array}{c} \text{derivation } D \\ \vdots \\ \hline \dots \quad (\text{fun}) \\ \dots \quad \dots <: \tau \quad (\text{sub}) \\ \hline \Gamma \vdash a : \tau \Rightarrow b' \end{array}$$

Montrer que $b' \approx b$.

Show that $b' \approx b$

Question 8

Même question si la dérivation se termine par une règle (sub) à gauche d'une règle (app) :

Same question as before when the derivation ends by a (sub) rule as left premise of an (app) rule:

$$\begin{array}{c} \text{derivation } D_1 \\ \vdots \\ \hline \dots <: \dots \quad (\text{sub}) \\ \dots \\ \hline \Gamma \vdash a : \tau \Rightarrow b \end{array} \quad \begin{array}{c} \text{derivation } D_2 \\ \vdots \\ \hline \dots \quad (\text{app}) \end{array}$$

Mettre cette dérivation sous la forme suivante :

Transform the derivation as follows:

$$\begin{array}{c} \text{derivation } D_1 \\ \vdots \quad \vdots \\ \hline \dots \quad (\text{app}) \\ \dots \quad \dots <: \tau \quad (\text{sub}) \\ \hline \Gamma \vdash a : \tau \Rightarrow b' \end{array}$$

et montrer que $b' \approx b$.

and show that $b' \approx b$.

Question 9

On appelle dérivation normalisée (notée ND) de $\Gamma \vdash a : \tau \Rightarrow b$ une dérivation dans laquelle la règle (sub) ne peut apparaître que juste au-dessus à droite de la règle (app), pour sous-typé le type de l'argument de la fonction, mais nulle part ailleurs :

A normal derivation (ND) for a judgment $\Gamma \vdash a : \tau \Rightarrow b$ is a derivation in which all occurrences of the rule (sub) are immediate right premises of (app) rules.

$$\begin{array}{c}
ND_2 \\
\vdots \\
\frac{ND_1 \quad \dots <: \dots}{\dots} \text{ (sub)} \\
\vdots \\
\frac{\dots}{\dots} \text{ (app)}
\end{array}$$

Montrer que si $\Gamma \vdash a : \tau \Rightarrow b$, alors il existe une dérivation de cet énoncé de la forme suivante : une dérivation normalisée suivie d'une application finale de la règle (sub).

Show that if $\Gamma \vdash a : \tau \Rightarrow b$ is derivable, then it can be derived by a normal derivation followed by a final application of the (sub) rule.

$$\frac{ND \quad \dots <: \tau}{\Gamma \vdash a : \tau \Rightarrow b'} \text{ (sub)}$$

Montrer que de plus $b \approx b'$.

Show that $b \approx b'$.

Question 10

Déduire de la question 9 que si $\Gamma \vdash a : \tau \Rightarrow b_1$ et $\Gamma \vdash a : \tau \Rightarrow b_2$, alors $b_1 \approx b_2$.

From question 9 deduce that $\Gamma \vdash a : \tau \Rightarrow b_1$ and $\Gamma \vdash a : \tau \Rightarrow b_2$, imply $b_1 \approx b_2$.

Algorithme de traduction / A translation algorithm

Question 11

Donner en pseudocode un algorithme $\mathcal{T}(\Gamma, a) = (\tau, b)$ qui prend en arguments un environnement de typage Γ et un terme a et retourne en résultats le type τ et le terme b tels que $\Gamma \vdash a : \tau \Rightarrow b$ par une dérivation normalisée, si elle existe, ou échoue sinon.

Write in pseudo-code an algorithm $\mathcal{T}(\Gamma, a) = (\tau, b)$ taking a type environment Γ and a term a and returning the type τ and term b such that $\Gamma \vdash a : \tau \Rightarrow b$ can be deduced by a normal derivation, if any, or failing otherwise.

Question 12

Dans le système de règles définissant $\Gamma \vdash a : \tau \Rightarrow b$ du préambule, on considère maintenant les conversions apparaissant dans b de manière symbolique : on ne remplace plus les termes $\mathcal{C}(\tau_1, \tau_2, b)$ par les λ -termes correspondants. D'après l'isomorphisme de Curry-Howard, tout terme b engendré par la grammaire

Consider again the deduction system for $\Gamma \vdash a : \tau \Rightarrow b$ given in the preamble but consider coercions occurring in b symbolically, that is, do not replace in them terms of the form $\mathcal{C}(\tau_1, \tau_2, b)$ by the corresponding λ -terms. According to the Curry-Howard isomorphism, every term b generated by the grammar

Termes de conversion / Coerce terms: $b ::= x \mid c \mid \lambda x:\tau.b \mid b_1 b_2 \mid \mathcal{C}(\tau_1, \tau_2, b)$

et produit par le système de règles du préambule encode une dérivation (une preuve) dans le λ -calcul simplement typé avec sous-typage (c.à.d. le système de typage-traduction du préambule sans les parties “ $\Rightarrow b$ ”). Dans les questions 7 et 8, on a montré comment réécrire les dérivations pour les mettre en forme normalisée. Réécrire une dérivation est isomorphe à réécrire le terme b qui encode cette dérivation.

1. Donner les règles de réécriture sur les termes de conversion b ci-dessus qui produit les termes encodant les dérivations normalisées. Expliquer pourquoi cela nécessite le typage explicite des b .
2. Modifier la grammaire des termes de conversion ci-dessus pour qu'elle produise uniquement les termes correspondant à des dérivations normalisées.

and deduced by the system in the preamble, encodes a proof/deduction in the simple type-system with subsumption (*ie*, the type system in the preamble in which all the $\Rightarrow b$ parts are removed). In Questions 7 and 8 you showed how to rewrite deductions to obtain normal derivations. Rewriting deductions is isomorphic to rewriting the terms that encode these deductions.

1. Give the rewriting rules on the coerce terms above that yield terms encoding normal derivations. Explain why this needs explicitly typed b terms.
2. Modify the grammar of coerce terms above so that it produces only terms corresponding to normal derivations.

Préservation sémantique / Semantic preservation

Nous étudions maintenant les propriétés sémantiques de la $\beta\eta$ -équivalence : précisément, nous nous demandons si lorsque $a \approx b$, les termes a et b s'exécutent avec le même comportement observable.

We now focus on semantic properties of $\beta\eta$ -equivalence: more precisely we study whether when $a \approx b$ it is the case that the terms a and b execute with the same observable behavior.

Question 13

On note $\xrightarrow{\beta\eta}$ une étape de β ou η réduction appliquée n'importe où dans un terme, y compris sous un λ :

We write $\xrightarrow{\beta\eta}$ for one step of β or η reduction, performed anywhere in a term, even under a λ :

$$C[(\lambda x : \tau. a) b] \xrightarrow{\beta\eta} C[a\{x \leftarrow b\}] \quad C[\lambda x : \tau. a x] \xrightarrow{\beta\eta} C[a] \quad (x \notin FV(a)) \quad \text{avec/with } C ::= [] \mid \lambda x : \tau. C \mid C a \mid a C$$

On rappelle que la réduction $\xrightarrow{\beta\eta}$ a la propriété de Church-Rosser :

Recall that the reduction $\xrightarrow{\beta\eta}$ enjoys the Church-Rosser property:

$$\text{Church-Rosser: } a \xrightarrow{\beta\eta}^* a_1 \wedge a \xrightarrow{\beta\eta}^* a_2 \implies \exists a', a_1 \xrightarrow{\beta\eta}^* a' \wedge a_2 \xrightarrow{\beta\eta}^* a'$$

Montrer que si $a_1 \approx a_2$, alors il existe a tel que $a_1 \xrightarrow{\beta\eta}^* a$ et $a_2 \xrightarrow{\beta\eta}^* a$.

Show that if $a_1 \approx a_2$, then there exists a such that $a_1 \xrightarrow{\beta\eta}^* a$ and $a_2 \xrightarrow{\beta\eta}^* a$.

Question 14

On note \xrightarrow{CBV} une étape de réduction en appel par valeur faible

$$C_{CBV}[(\lambda x : \tau. a) v] \xrightarrow{CBV} C_{CBV}[a\{x \leftarrow v\}]$$

Au contraire de $\xrightarrow{\beta\eta}$, la réduction \xrightarrow{CBV} ne s'applique pas sous les λ (réduction faible) et ne fait pas d' η -réductions. La réduction \xrightarrow{CBV} modélise l'exécution d'un programme, alors que $\xrightarrow{\beta\eta}$ modélise les simplifications qu'un compilateur effectue sur le programme.

Soit τ un type de base (`int` ou `float`) et a_1, a_2 deux termes tels que

$$a_1 \approx a_2 \quad \emptyset \vdash a_1 : \tau \quad \emptyset \vdash a_2 : \tau$$

À l'aide du résultat de la question 13, montrer que a_1 et a_2 s'évaluent en appel par valeur en une même constante c :

$$\exists c, a_1 \xrightarrow{CBV}^* c \wedge a_2 \xrightarrow{CBV}^* c$$

En conclure que toutes les traductions b d'un terme a au type τ ($\emptyset \vdash a : \tau \Rightarrow b$) ont le même comportement observable.

Question 15

Si on ajoute au langage des fonctions récursives (p.ex. un combinateur de point fixe), les résultats de la question 14 restent-ils vrais? Que pourrait-on changer dans la définition de la traduction pour que toutes les traductions d'un terme divergent divergent?

We write \xrightarrow{CBV} for one step of reduction in weak call-by-value:

$$\text{avec/with } C_{CBV} ::= [] \mid C_{CBV} a \mid v C_{CBV}$$

Unlike $\xrightarrow{\beta\eta}$, the \xrightarrow{CBV} reduction does not apply under λ (weak reduction) and does not perform η -reductions. The \xrightarrow{CBV} reduction models the execution of a program, while $\xrightarrow{\beta\eta}$ models compile-time simplifications over a program.

Let τ be a base type (`int` or `float`) and a_1, a_2 two terms such that

Using the result of question 13, show that a_1 and a_2 evaluate (in call-by-value) to some constant c :

Conclude that all translations b of a term a at type τ ($\emptyset \vdash a : \tau \Rightarrow b$) have the same observable behavior.

If we add recursive functions (for instance, a fix-point combinator) to our language, does the results of question 14 still hold? How could we modify the definition of the translation so that all translations of a diverging term diverge?

Corrigé / Answers

Question 1

$$\frac{\frac{\text{cos} : \text{float} \rightarrow \text{float}}{\Gamma \vdash \text{cos} : \text{float} \rightarrow \text{float} \Rightarrow \text{cos}} \text{ (const)} \quad \frac{\frac{1 : \text{int}}{\Gamma \vdash 1 : \text{int} \Rightarrow 1} \text{ (const)} \quad \text{int} <: \text{float}}{\Gamma \vdash 1 : \text{float} \Rightarrow : \text{floatofint1}} \text{ (sub)}}{\Gamma \vdash \text{cos}1 : \text{float} \Rightarrow \text{cos}(\text{floatofint1})} \text{ (app)}$$

see also the answer to question 5 for a different solution

Question 2

$$\mathcal{C}(\sigma \rightarrow \tau, \sigma' \rightarrow \tau', a) = \lambda x : \sigma'. \mathcal{C}(\tau, \tau', a \mathcal{C}(\sigma', \sigma, x))$$

Question 3

$$\begin{aligned} \mathcal{C}(\tau_1 \times \tau_2, \tau_1, a) &= \pi_1 a \\ \mathcal{C}(\tau_1 \times \tau_2, \tau_2, a) &= \pi_2 a \\ \mathcal{C}(\tau_1 \times \tau_2 \rightarrow \tau_3, \tau_1 \rightarrow \tau_2 \rightarrow \tau_3, a) &= \lambda x : \tau_1. \lambda y : \tau_2. a(\text{pair } x y) \\ \mathcal{C}(\tau_1 \rightarrow \tau_2 \rightarrow \tau_3, \tau_1 \times \tau_2 \rightarrow \tau_3, a) &= \lambda x : \tau_1 \times \tau_2. a(\pi_1 x)(\pi_2 x) \\ \mathcal{C}(\tau_1 \rightarrow \tau_2 \times \tau_3, (\tau_1 \rightarrow \tau_2) \times (\tau_1 \rightarrow \tau_3), a) &= \text{pair}(\lambda x : \tau_1. \pi_1(a x))(\lambda x : \tau_1. \pi_2(a x)) \\ \mathcal{C}((\tau_1 \rightarrow \tau_2) \times (\tau_1 \rightarrow \tau_3), \tau_1 \rightarrow \tau_2 \times \tau_3, a) &= \lambda x : \tau_1. \text{pair}((\pi_1 a)x)((\pi_2 a)x) \\ \mathcal{C}(\sigma_1 \times \sigma_2, \tau_1 \times \tau_2, a) &= \text{pair}(\mathcal{C}(\sigma_1, \tau_1, \pi_1 a))(\mathcal{C}(\sigma_2, \tau_2, \pi_2 a)) \end{aligned}$$

Question 4

First show that $\Gamma \vdash \mathcal{C}(\tau_1, \tau_2, b) : \tau_2$ whenever $\Gamma \vdash b : \tau_1$ and $\tau_1 <: \tau_2$. (Induction on the derivation of $\tau_1 <: \tau_2$.)

The result then follows from an induction on the derivation of $\Gamma \vdash a : \tau \Rightarrow b$ and case analysis on the last rule used, using the lemma above in the (sub) case.

Question 5

Instead of coercing 1 from `int` to `float`, we can coerce `cos` from `float` \rightarrow `float` to `int` \rightarrow `float`.

$$\frac{\frac{\text{cos} : \text{float} \rightarrow \text{float}}{\Gamma \vdash \text{cos} : \text{float} \rightarrow \text{float} \Rightarrow \text{cos}} \text{ (const)} \quad \frac{\text{float} \rightarrow \text{float} <: \text{int} \rightarrow \text{float}}{\Gamma \vdash \text{cos} : \text{int} \rightarrow \text{float} \Rightarrow \mathcal{C}(\text{float} \rightarrow \text{float}, \text{int} \rightarrow \text{float}, \text{cos})} \text{ (sub)} \quad \frac{1 : \text{int}}{\Gamma \vdash 1 : \text{int} \Rightarrow 1} \text{ (const)}}{\Gamma \vdash \text{cos } 1 : \text{float} \Rightarrow \mathcal{C}(\text{float} \rightarrow \text{float}, \text{int} \rightarrow \text{float}, \text{cos}) } 1} \text{ (app)}$$

The translation is $\mathcal{C}(\text{float} \rightarrow \text{float}, \text{int} \rightarrow \text{float}, \text{cos}) 1 = (\lambda x : \text{int}. \text{cos}(\text{floatofint } x)) 1$, which is different from the translation $\text{cos}(\text{floatofint } 1)$ obtained previously, but the former β -reduces to the latter.

Question 6

First property: $\forall \tau, a, \mathcal{C}(\tau, \tau, a) \approx a$. Obvious if $\tau = \mathbf{int}$ or $\tau = \mathbf{float}$. If $\tau = \tau_1 \rightarrow \tau_2$, by induction hypothesis $\mathcal{C}(\tau_1, \tau_1, x) \approx x$ and $\mathcal{C}(\tau_2, \tau_2, a \mathcal{C}(\tau_1, \tau_1, x)) \approx a \mathcal{C}(\tau_1, \tau_1, x)$. Therefore,

$$\mathcal{C}(\tau, \tau, a) = \lambda x:\tau_1. \mathcal{C}(\tau_2, \tau_2, a \mathcal{C}(\tau_1, \tau_1, x)) \approx \lambda x:\tau_1. a \ x \approx a \quad \text{by } \eta$$

Second property: $\forall \tau_1, \tau_2, \tau_3, a, \tau_1 <: \tau_2 \wedge \tau_2 <: \tau_3 \implies \mathcal{C}(\tau_1, \tau_3, a) \approx \mathcal{C}(\tau_2, \tau_3, \mathcal{C}(\tau_1, \tau_2, a))$.

Base cases: all 3 types are base types ($\mathbf{int/int/int}$ or $\mathbf{int/int/float}$ or $\mathbf{int/float/float}$ or $\mathbf{float/float/float}$) and the result is trivial.

Inductive case: $\sigma_1 \rightarrow \tau_1 <: \sigma_2 \rightarrow \tau_2 <: \sigma_3 \rightarrow \tau_3$. We have $\sigma_3 <: \sigma_2 <: \sigma_1$ and $\tau_1 <: \tau_2 <: \tau_3$, and by induction hypothesis:

$$\begin{aligned} \mathcal{C}(\sigma_2 \rightarrow \tau_2, \sigma_3 \rightarrow \tau_3, \mathcal{C}(\sigma_1 \rightarrow \tau_1, \sigma_2 \rightarrow \tau_2, a)) &= \lambda x:\sigma_3. \mathcal{C}(\tau_2, \tau_3, (\lambda y:\sigma_2. \mathcal{C}(\tau_1, \tau_2, a \mathcal{C}(\sigma_2, \sigma_1, y)))) \mathcal{C}(\sigma_3, \sigma_2, x)) \\ &\approx \lambda x:\sigma_3. \mathcal{C}(\tau_2, \tau_3, \mathcal{C}(\tau_1, \tau_2, a \mathcal{C}(\sigma_2, \sigma_1, \mathcal{C}(\sigma_3, \sigma_2, x)))) \quad (\beta) \\ &\approx \lambda x:\sigma_3. \mathcal{C}(\tau_1, \tau_3, a \mathcal{C}(\sigma_3, \sigma_1, x)) \quad (\text{ind. hyps.}) \\ &= \mathcal{C}(\sigma_1 \rightarrow \tau_1, \sigma_3 \rightarrow \tau_3, a) \end{aligned}$$

Question 7

The initial derivation is:

$$\frac{\frac{\Gamma; x : \tau' \vdash a : \sigma \Rightarrow b \quad \sigma <: \tau}{\Gamma; x : \tau' \vdash a : \tau \Rightarrow \mathcal{C}(\sigma, \tau, b)}}{\Gamma \vdash \lambda x : \tau'. a : \tau' \rightarrow \tau \Rightarrow \lambda x : \tau'. \mathcal{C}(\sigma, \tau, b)}$$

Since $\sigma <: \tau$, we have $\tau' \rightarrow \sigma <: \tau' \rightarrow \tau$. Hence the following derivation:

$$\frac{\frac{\Gamma; x : \tau' \vdash a : \sigma \Rightarrow b}{\Gamma \vdash \lambda x:\tau'. a : \tau' \rightarrow \sigma \Rightarrow \lambda x:\tau'. b} \quad \tau' \rightarrow \sigma <: \tau' \rightarrow \tau}{\Gamma \vdash \lambda x : \tau'. a : \tau' \rightarrow \tau \Rightarrow \mathcal{C}(\tau' \rightarrow \sigma, \tau' \rightarrow \tau, \lambda x:\tau'. b)}$$

We have $b' = \mathcal{C}(\tau' \rightarrow \sigma, \tau' \rightarrow \tau, \lambda x:\tau'. b) = \lambda x : \tau'. \mathcal{C}(\sigma, \tau, (\lambda x:\tau'. b) \mathcal{C}(\tau', \tau', x)) \approx \lambda x : \tau'. \mathcal{C}(\sigma, \tau, b)$ by question 6 (first property) and β .

Question 8

The initial derivation is:

$$\frac{\frac{\Gamma \vdash a_1 : \sigma' \rightarrow \sigma \Rightarrow b_1 \quad \sigma' \rightarrow \sigma <: \tau' \rightarrow \tau}{\Gamma \vdash a_1 : \tau' \rightarrow \tau \Rightarrow \mathcal{C}(\sigma' \rightarrow \sigma, \tau' \rightarrow \tau, b_1)} \quad \Gamma \vdash a_2 : \tau' \Rightarrow b_2}{\Gamma \vdash a_1 a_2 : \tau \Rightarrow \mathcal{C}(\sigma' \rightarrow \sigma, \tau' \rightarrow \tau, b_1) b_2}$$

Since $\sigma' \rightarrow \sigma <: \tau' \rightarrow \tau$, we have $\tau' <: \sigma'$ and $\sigma <: \tau$. Hence the following derivation:

$$\frac{\frac{\Gamma \vdash a_1 : \sigma' \rightarrow \sigma \Rightarrow b_1 \quad \frac{\Gamma \vdash a_2 : \tau' \Rightarrow b_2 \quad \tau' <: \sigma'}{\Gamma \vdash a_2 : \sigma' \Rightarrow \mathcal{C}(\tau', \sigma', b_2)}}{\Gamma \vdash a_1 a_2 : \sigma \Rightarrow b_1 \mathcal{C}(\tau', \sigma', b_2)} \quad \sigma <: \tau}{\Gamma \vdash a_1 a_2 : \tau \Rightarrow \mathcal{C}(\sigma, \tau, b_1 \mathcal{C}(\tau', \sigma', b_2))}$$

Therefore $b' = \mathcal{C}(\sigma, \tau, b_1 \mathcal{C}(\tau', \sigma', b_2))$. Moreover,

$$\begin{aligned} \mathcal{C}(\sigma' \rightarrow \sigma, \tau' \rightarrow \tau, b_1) b_2 &= (\lambda x : \tau'. \mathcal{C}(\sigma, \tau, b_1 \mathcal{C}(\tau', \sigma', x))) b_2 \\ &\approx \mathcal{C}(\sigma, \tau, b_1 \mathcal{C}(\tau', \sigma', b_2)) \\ &= b' \end{aligned}$$

Question 9

By induction on the (not normal) derivation of $\Gamma \vdash a : \tau \Rightarrow b$ and by case analysis on the last rule used.

Axioms (var) and (const): the derivation is already normalized; take $b' = b$ and $\tau' = \tau$.

Rule (fun): the initial derivation is of the form

$$\frac{\begin{array}{c} \vdots \\ \Gamma_1 \vdash a_1 : \tau_1 \Rightarrow b_1 \end{array}}{\Gamma \vdash a : \tau \Rightarrow b} \text{ (fun)}$$

with $a = \lambda x : \tau'. a_1$, etc. Applying the induction hypothesis to the premise, we get a derivation of the form:

$$\frac{\begin{array}{c} ND \\ \vdots \\ \Gamma_1 \vdash a_1 : \tau_1 \Rightarrow b_2 \end{array}}{\Gamma_1 \vdash a_1 : \tau_1 \Rightarrow b_2} \text{ (sub)}$$

with $b_2 \approx b_1$. Applying rule (fun) to the result, we obtain:

$$\frac{\begin{array}{c} ND \\ \vdots \\ \Gamma_1 \vdash a_1 : \tau_1 \Rightarrow b_2 \end{array}}{\Gamma_1 \vdash a_1 : \tau_1 \Rightarrow b_2} \text{ (sub)} \\ \frac{\Gamma_1 \vdash a_1 : \tau_1 \Rightarrow b_2}{\Gamma \vdash a : \tau \Rightarrow b_3} \text{ (fun)}$$

with $b_3 = \lambda x. b_2 \approx \lambda x. b_1 = b$. Permuting (sub) et (fun) as in question 7, we obtain:

$$\frac{\begin{array}{c} ND \\ \vdots \\ \Gamma \vdash a : \tau \Rightarrow b_4 \end{array}}{\Gamma \vdash a : \tau \Rightarrow b_4} \text{ (fun)}$$

with $b_4 \approx b_3$. The sub-derivation $ND + (fun)$ being normalized, this derivation is what we want, with $b' = b_4 \approx b$ by transitivity.

Rule (app): applying the induction hypothesis to the two premises, and recombining the results with (app), we obtain:

$$\frac{\begin{array}{c} ND_1 \\ \vdots \\ \Gamma \vdash a : \tau \Rightarrow b_1 \end{array} \quad \begin{array}{c} ND_2 \\ \vdots \\ \Gamma \vdash a : \tau \Rightarrow b_1 \end{array}}{\Gamma \vdash a : \tau \Rightarrow b_1} \text{ (app)}$$

with $b_1 \approx b$. Permuting (app) and the left (sub) as in question 8, we get:

$$\begin{array}{c}
 ND_2 \\
 \vdots \\
 \hline \dots \text{ (sub)} \\
 ND_1 \quad \hline \dots \text{ (sub)} \\
 \vdots \\
 \hline \dots \text{ (app)} \\
 \hline \dots \text{ (sub)} \\
 \hline \Gamma \vdash a : \tau \Rightarrow b_2
 \end{array}$$

with $b_2 \approx b_1$. By transitivity (question 4), we can combine both subsumption steps to the right, obtaining:

$$\begin{array}{c}
 ND_2 \\
 \vdots \\
 \hline \dots \text{ (sub)} \\
 ND_1 \quad \hline \dots \text{ (app)} \\
 \vdots \\
 \hline \dots \text{ (sub)} \\
 \hline \Gamma \vdash a : \tau \Rightarrow b_3
 \end{array}$$

with $b_3 \approx b_2$. This derivation is of the desired shape: a normalized derivation + a (sub) step.

Rule (sub): by induction hypothesis,

$$\begin{array}{c}
 ND \\
 \vdots \\
 \hline \dots \text{ (sub)} \\
 \hline \dots \text{ (sub)} \\
 \hline \Gamma \vdash a : \tau \Rightarrow b_1
 \end{array}$$

with $b_1 \approx b$. We then combine the two (sub) steps into one using question 4 and obtain a derivation of the expected shape.

Question 10

Applying question 9 to the two derivations of $\Gamma \vdash a : \tau \Rightarrow b_1$ and $\Gamma \vdash a : \tau \Rightarrow b_2$, we obtain derivations of the shape:

$$\begin{array}{c}
 ND_1 \\
 \vdots \\
 \hline \Gamma \vdash a : \tau_1 \Rightarrow b'_1 \quad \tau_1 <: \tau \\
 \hline \Gamma \vdash a : \tau \Rightarrow \mathcal{C}(\tau_1, \tau, b'_1)
 \end{array}
 \qquad
 \begin{array}{c}
 ND_2 \\
 \vdots \\
 \hline \Gamma \vdash a : \tau_2 \Rightarrow b'_2 \quad \tau_2 <: \tau \\
 \hline \Gamma \vdash a : \tau \Rightarrow \mathcal{C}(\tau_2, \tau, b'_2)
 \end{array}$$

Note that normal derivations are *syntax-directed* and therefore uniquely determined by Γ and a . It follows that $ND_1 = ND_2$, and therefore $\tau_1 = \tau_2$ and $b'_1 = b'_2$. Since $b_1 \approx \mathcal{C}(\tau_1, \tau, b'_1)$ et $b_2 \approx \mathcal{C}(\tau_2, \tau, b'_2)$, it follows $b_1 \approx b_2$.

Question 11

```

 $\mathcal{T}(\Gamma, a) = \text{match } a \text{ with}$ 
   $x \rightarrow \text{if } x \in \text{dom}(\Gamma) \text{ then } (\Gamma(x), x) \text{ else fail}$ 
   $| c \rightarrow \text{if } c : \tau \text{ then } (\tau, c)$ 
   $| \lambda x : \tau. a \rightarrow \text{let } (\sigma, b) = \mathcal{T}(\Gamma + \{x : \tau\}, a) \text{ in } (\tau \rightarrow \sigma, \lambda x : \tau. b)$ 
   $| a_1 a_2 \rightarrow \text{let } (\tau' \rightarrow \tau, b_1) = \mathcal{T}(\Gamma, a_1) \text{ in}$ 
     $\text{let } (\tau'', b_2) = \mathcal{T}(\Gamma, a_2) \text{ in}$ 
     $\text{if } \mathcal{S}(\tau'', \tau') \text{ then } (\tau, b_1 \mathcal{C}(\tau'', \tau', b_2)) \text{ else fail}$ 

```

Question 12

1. It suffices to add a rewriting rule $b \rightsquigarrow b'$ for the pairs of terms b, b' found in Questions 7 and 8, plus a rule for eliminating two consecutive applications of subsumption, namely:

$$\begin{aligned}
 \lambda x : \tau'. \mathcal{C}(\sigma, \tau, b) &\rightsquigarrow \mathcal{C}(\tau' \rightarrow \sigma, \tau' \rightarrow \tau, \lambda x : \tau'. b) \\
 \mathcal{C}(\sigma' \rightarrow \sigma, \tau' \rightarrow \tau, b_1) b_2 &\rightsquigarrow \mathcal{C}(\sigma, \tau, b_1 \mathcal{C}(\tau', \sigma', b_2)) \\
 \mathcal{C}(\tau_2, \tau_3, \mathcal{C}(\tau_1, \tau_2, b)) &\rightsquigarrow \mathcal{C}(\tau_1, \tau_3, b)
 \end{aligned}$$

The explicit typing is needed to determine the τ' in the first rewriting rule.

2. Normal Coerce Terms $b ::= x \mid c \mid \lambda x. b \mid b_1 b_2 \mid b_1 \mathcal{C}(\tau_1, \tau_2, b_2)$

Question 13

By induction on a derivation of $a_1 \approx a_2$ and case analysis on the last rule used:

- Cases (β) and (η): take $a = a_2$.
- Case (equiv-abstr): by induction hypothesis, $a_1 \xrightarrow{\beta\eta}^* a \xleftarrow{\beta\eta}^* a_2$, therefore $\lambda x. a_1 \xrightarrow{\beta\eta}^* \lambda x. a \xleftarrow{\beta\eta}^* \lambda x. a_2$.
- Case (equiv-app): as in the previous case.
- Cases (equiv-refl) and (equiv-sym): obvious.
- Case (equiv-trans): by induction hypothesis, $a_1 \xrightarrow{\beta\eta}^* a' \xleftarrow{\beta\eta}^* a_2$ and $a_2 \xrightarrow{\beta\eta}^* a'' \xleftarrow{\beta\eta}^* a_3$. By the Church-Rosser theorem, there exists a such that $a' \xrightarrow{\beta\eta}^* a \xleftarrow{\beta\eta}^* a''$. Therefore, $a_1 \xrightarrow{\beta\eta}^* a \xleftarrow{\beta\eta}^* a_3$.

Question 14

The simply-typed λ -calculus is normalizing. Therefore, there exists CBV normal forms n_1, n_2 such that

$$a_1 \xrightarrow{CBV}^* n_1 \quad a_2 \xrightarrow{CBV}^* n_2$$

By subject reduction and progress, n_1 and n_2 are values of type τ ($= \text{int}$ or float). Therefore, they cannot be λ -abstractions and must be constants c_1, c_2 :

$$a_1 \xrightarrow{CBV}^* c_1 \quad a_2 \xrightarrow{CBV}^* c_2$$

Now, $a_1 \approx a_2$, therefore $a_1 \xrightarrow{\beta\eta}^* a \xleftarrow{\beta\eta}^* a_3$ for some a . Moreover, CBV reductions are a special case of $\beta\eta$ reductions: $\xrightarrow{CBV} \subset \xrightarrow{\beta\eta}$. It follows that $a_1 \xrightarrow{\beta\eta}^* c_1$ and $a_2 \xrightarrow{\beta\eta}^* c_2$. Applying the Church-Rosser theorem 3 times,

it follows that $c_1 \xrightarrow[\beta\eta]{*} d \xleftarrow[\beta\eta]{*} c_2$ for some term d . Since constants do not reduce, $c_1 = d = c_2$ and the expected result follows.

For the corollary, assume $\emptyset \vdash a : \tau \Rightarrow b_1$ and $\emptyset \vdash a : \tau \Rightarrow b_2$. We have $b_1 \approx b_2$ by question 10, and we just showed that exists a constant c such that b_1 and b_2 evaluate to c . The two translations terminate on the same value: therefore they have the same observable behavior.

Question 15

Recursion brings the possibility of divergence, and divergence invalidates the results of question 14. Consider a diverging term $\omega = \mathbf{let\ rec\ } f = \lambda x:\mathbf{int}. f\ x\ \mathbf{in\ } f$. Without using subsumption, we can derive $\emptyset \vdash \omega : \mathbf{int} \rightarrow \mathbf{int} \Rightarrow \omega$. However, by adding a final, trivial step of subsumption $\mathbf{int} \rightarrow \mathbf{int} <: \mathbf{int} \rightarrow \mathbf{int}$, we can also translate ω to

$$b = \mathcal{C}(\mathbf{int} \rightarrow \mathbf{int}, \mathbf{int} \rightarrow \mathbf{int}, \omega) = \lambda x:\mathbf{int}. \omega\ x$$

This translation b terminates while the ω translation does not.

The major offender here is the definition of coercion over function types:

$$\mathcal{C}(\sigma \rightarrow \tau, \sigma' \rightarrow \tau', a) = \lambda x:\sigma'. \mathcal{C}(\tau, \tau', a\ \mathcal{C}(\sigma', \sigma, x))$$

which “hides” immediate divergence of a by putting it under a λ . One possible fix is to **let**-bind a explicitly:

$$\mathcal{C}(\sigma \rightarrow \tau, \sigma' \rightarrow \tau', a) = \mathbf{let\ } f = a\ \mathbf{in\ } \lambda x:\sigma'. \mathcal{C}(\tau, \tau', f\ \mathcal{C}(\sigma', \sigma, x))$$

In the definition of $\beta\eta$ -equivalence, we can then replace the η case

$$(\lambda x. a\ x) \approx a \quad (x \notin FV(a))$$

(which does not preserve divergence) with the following two cases

$$(\lambda x. v\ x) \approx a \quad (v \text{ value or variable, } v \neq x) \quad \mathbf{let\ } x = a\ \mathbf{in\ } x \approx a$$

which both preserve divergence.