Rich types, Tractable typing – Subtyping –

Yann Régis-Gianas
yrg@irif.fr

2017-12-22

Yann Régis-Gianas
yrg@irif.fr

What is this document?

Here are the answers to the exercises described in

slides/02-yrg-diy-subtyping.pdf in the course repository.

Plan

$\lambda_{<:}$

Let us consider the following syntax for terms of a λ-calculus with records, that we will call $\lambda_{<:}$:

$$
\begin{array}{rcl}
t & ::= & x \\
 & | & 0, 1, \ldots \\
 & | & t\,u \\
 & | & \lambda(x : \tau).t \\
 & | & \{(\ell_i = t_i)_{i \in [1..n]}\} \\
 & | & t.\ell
\end{array}
\qquad
\begin{array}{rcl}
\tau & ::= & \top \\
 & | & \mathsf{nat} \\
 & | & \tau \rightarrow \tau \\
 & | & \{(\ell_i : \tau_i)_{i \in [1..n]}\}
\end{array}
$$

where $x, y, \ldots$ denote identifiers taken in some enumerable set $\mathcal{I}$ and where $\ell, \ell_1, \ell_2, \ldots$ denote labels taken in some enumerable set $\mathcal{L}$. We want to equip this language with a call-by-value weak reduction strategy.

Values for $\lambda_{<:}$

Question 1.1

Define the syntax for values v.

Values for $\lambda_{<:}$

Question 1.1

Define the syntax for values v.

Answer

$$\begin{array}{rcl} v & ::= & x \\ & \mid & 0, 1, \ldots \\ & \mid & \lambda x.t \\ & \mid & \{(\ell_i = v_i)_{i \in [1..n]}\} \end{array}$$

Implicit invariants of the syntax

When we write:

$$(\ell_i : \tau_i)_{i \in [1..n]}$$

or

$$(\ell_i = t_i)_{i \in [1..n]}$$

or

$$(\ell_i = v_i)_{i \in [1..n]}$$

we implicitly assume that $\forall i \neq j, \ell_i \neq \ell_j$.

Evaluation contexts for $\lambda_{<:}$

Question 1.2

Define a syntax for contexts C that encode the reduction strategy we are targetting.

Yann Régis-Gianas
yrg@irif.fr

Evaluation contexts for $\lambda_{<:}$

Question 1.2

Define a syntax for contexts C that encode the reduction strategy we are targetting.

Answer

The following syntax for evaluation contexts encodes a call-by-value weak reduction:

$$\begin{array}{rcl} C & ::= & [] \\ & \mid & C\,t \\ & \mid & v\,C \\ & \mid & \{fvs; \ell = C; fts\} \\ & \mid & C.\ell \\ fvs & ::= & \bullet \mid fvs; \ell = v \\ fts & ::= & \bullet \mid fts; \ell = t \end{array}$$

Notice that Didier Rémy uses a notion of elementary evaluation contexts in this course note. For such contexts, every occurrence of C is replaced by $[]$ on the right hand side of grammar rules.

Reduction rules

Question 1.3

What should be the reduction rules for this language?

Reduction rules

Question 1.3

What should be the reduction rules for this language?

Answer

$$\begin{array}{rcll} C[t] & \rightarrow & C[u] & \text{where } t \rightarrow u \text{ and } C \neq [] \\ (\lambda x.t)v & \rightarrow & t[x \mapsto v] & \\ \{(\ell_i = v_i)_{i \in [1..n]}\}.\ell_j & \rightarrow & v_j & \text{where } j \in [1..n] \end{array}$$

Unique decomposition

Question 1.4

Prove that for every reducible term t, there is a unique decomposition of the form $t = C[u]$ so that u is a β-redex or a field projection redex.

Unique decomposition

Question 1.4

Prove that for every reducible term t, there is a unique decomposition of the form $t = C[u]$ so that u is a β-redex or a field projection redex.

Answer

Formally, we ought to prove that:

Lemma

For every term t such that $t \rightarrow u$ then there exists a unique C such that $t = C[t']$.

Unique decomposition (proof)

Yann Régis-Gianas
yrg@irif.fr

Proof.

By induction over the reducible terms t and by case on the last rule used to derive $t \rightarrow u$. If t and u are related by a β-reduction or a field access, then C is $[]$ and no other context works. Otherwise, let us assume that:

$$t = C[u_0] \rightarrow C[u_0'] \quad \text{where } u_0 \rightarrow u_0'$$

and

$$t = C'[u_1] \rightarrow C'[u_1'] \quad \text{where } u_1 \rightarrow u_1'$$

Assume that $C = C_0\ t_1$. Then, $C_0[u_0] \rightarrow C_0[u_0']$ (**1**). Necessarily, t is an application of the form $t_0\ t_1$ with $t_0 = C_0[u_0]$. Hence, C' is of the form $C_1\ t_1$ or $v\ C_1$. The second form contradicts (1) since a value is irreducible. By induction hypothesis applied to t_0 (which is reducible because of (1)), we get that $C_0 = C_1$, thus $C = C'$.
The other cases are similar. □

Subsomption

Consider the following function:

$$\lambda(r : \{\ell : \tau\}).r.\ell$$

, it can be safely applied to every record that contains at least a field ℓ of type τ. In simply-typed λ-calculus, the valid arguments for this function are the records that contain exactly one field ℓ of type τ. Hence, STLC is strict restriction of what can be safely allowed. The purpose of the subtyping relation is to remove this restriction.

We introduce a relation written "$\tau <: \tau'$" which is read "τ is a subtype of τ'. Intuitively, if $\tau <: \tau'$ holds, then every term of type τ can be used where a term of type τ' is valid. This is formally captured by the following (Subsomption) rule:

Subsomption

$$\frac{\Gamma \vdash t : \tau \qquad \tau <: \tau'}{\Gamma \vdash t : \tau'}$$

Typing rules

Yann Régis-Gianas
yrg@irif.fr

Question 2.1

Give the two typing rules (Record) and (Field).

Answer

Field

$$\frac{\Gamma \vdash t : \{\ell_i : \tau_i\}_{i \in [1..n]} \qquad j \in [1..n]}{\Gamma \vdash t.\ell_j : \tau_j}$$

Record

$$\frac{\forall i \in [1..n] \quad \Gamma \vdash t_i : \tau_i}{\Gamma \vdash \{\ell_i = t_i\}_{i \in [1..n]} : \{\ell_i : \tau_i\}_{i \in [1..n]}}$$

The case for functions

Question 2.2

Assume $\mathsf{succ} : \mathsf{nat} \rightarrow \mathsf{nat}$.
Consider the function

$$(\lambda(f :?).\mathsf{succ}((f\{\ell_1 = 31; \ell_2 = 73\}).\ell_3))$$

, can you characterize the types that make valid ascriptions for f?

Answer

The function f must be able to accept any record that has at most two integer fields ℓ_1 and ℓ_2. It must produce a record that has at least an integer field ℓ_3. Therefore, any type ascription of the form $\tau_1 \rightarrow \tau_2$ such that $\{\ell_1 : \mathbf{nat}; \ell_2 : \mathbf{nat}\} <: \tau_1$ and $\tau_2 <: \{\ell_3 : \mathbf{nat}\}$.

Subtyping relation

Question 2.3

Propose a set of rules to define the subtyping relation $\tau <: \tau'$.

Subtyping relation

Answer

$$\frac{}{\tau <: \top}\ \text{Top}$$

Subtyping relation

Answer

Reflexivity

$$\frac{}{\tau <: \tau}$$

Transitivity

$$\frac{\tau_1 <: \tau_2 \qquad \tau_2 <: \tau_3}{\tau_1 <: \tau_3}$$

Subtyping relation

Answer

Sub-Arrow

$$\frac{\tau_1' <: \tau_1 \qquad \tau_2 <: \tau_2'}{\tau_1 \rightarrow \tau_2 <: \tau_1' \rightarrow \tau_2'}$$

Notice that the arrow type constructor is **contravariant** in its first argument (on inputs) while it is **covariant** in its second argument (on outputs).

Subtyping relation

Yann Régis-Gianas
yrg@irif.fr

Answer

Sub-Record-Width

$$\frac{n < m}{\{(\ell_i : \tau_i)_{i \in [1..m]}\} <: \{(\ell_j : \tau_j)_{j \in [1..n]}\}}$$

Subtyping relation

Answer

Sub-Record-Permutation

$$\frac{\sigma \text{ is a permutation of } [1..n] \text{ which is not the identity} \qquad \forall i \quad \ell_i = \ell'_{\sigma(i)} \qquad \tau_i = \tau'_{\sigma(i)}}{\{(\ell_i : \tau_i)_{i \in [1..n]}\} <: \{(\ell'_j : \tau'_j)_{j \in [1..n]}\}}$$

The only operator on records is field projection and this operator is not sensitive to the order of appearance of the fields in the record. Therefore, any permutation of the fields can be applied to a record without changing its observational behavior.

Subtyping relation

Answer

Sub-Record-Depth

$$\frac{\forall i \quad \tau_i <: \tau_i'}{\{(\ell_i : \tau_i)_{i \in [1..n]}\} <: \{(\ell_i : \tau_i')_{i \in [1..n]}\}}$$

Record types are therefore covariant wrt to the types of their fields.

Subtyping relation

Question 2.4

Is the resulting relation a partial order?

Subtyping relation

Question 2.4

Is the resulting relation a partial order?

Answer

No! This relation is not antireflexive. Take $\tau_1 = \{x : \textbf{nat} : y : \textbf{nat}\}$ and $\tau_2 = \{y : \textbf{nat} : x : \textbf{nat}\}$. We have $\tau_1 <: \tau_2$ and $\tau_2 <: \tau_1$ but these are two distinct types.

Type soundness : Subject Reduction

Question 2.5

State and prove the “Subject Reduction” property. It is recommended to decompose the proof through the introduction of the usual Lemmas about inversions of judgements and substitutions.

Type soundness : Subject Reduction

Question 2.5

State and prove the “Subject Reduction” property. It is recommended to decompose the proof through the introduction of the usual Lemmas about inversions of judgements and substitutions.

Answer

It is a good exercise to do this proof carefully without forgetting any details. This proof needs:

Inversion Lemmas about the subtyping relation.

Inversion Lemmas about the typing judgment.

A substitution Lemma.

Inversion Lemmas about the subtyping relation

Lemma (Inversion of a subtyping relation between arrows)

If $\tau <: \tau_1 \rightarrow \tau_2$ holds then there exists τ_1' and τ_2' such that $\tau_1 <: \tau_1'$ and $\tau_2 <: \tau_2'$.

Proof.

By induction on the size of the derivation and by cases on the last rule used to conclude $\tau <: \tau_1 \rightarrow \tau_2$.

- ▷ The last rule is (Sub-Arrow). In that case, the Lemma holds immediately.
- ▷ The last rule is (Sub-Reflexive). In that case, $\tau = \tau_1 \rightarrow \tau_2$ and we choose $\tau_1' = \tau_1$ and $\tau_2 = \tau_2'$. By reflexivity, we get the conclusion.
- ▷ The last rule is (Sub-Transitivity). In that case, there exists τ' such that $\tau <: \tau'$ (**1**) and $\tau' <: \tau_1 \rightarrow \tau_2$ (**2**). By induction hypothesis applied on (2), there exists τ_1' and τ_2' such that $\tau' = \tau_1' \rightarrow \tau_2'$ with $\tau_1 <: \tau_1'$ (**3**) and $\tau_2' <: \tau_2$ (**4**). We apply the induction hypothesis a second time on (1). This leads to the existence of τ_1'' and τ_2'' such that $\tau_1' <: \tau_1''$ (**5**) and $\tau_2'' <: \tau_2'$ (**6**). By (Sub-Transitivity) applied to (3) and (5), we get $\tau_1 <: \tau_1''$. By (Sub-Transitivity) applied to (4) and (6), we get $\tau_2'' <: \tau_2$.

□

Inversion Lemmas about the subtyping relation

Lemma (Inversion of the subtyping relation between records)

If $\tau <: \{(\ell'_j : \tau'_j)_{j\in[1..m]}\}$ holds, then $\tau = \{(\ell_i : \tau_i)_{i\in[1..n]}\}$ and there exists an injective function ϕ from $[1..n]$ to $[1..m]$ such that forall $i \in [1..n]$, $\ell_i = \ell'_{\phi(i)}$ and $\tau_i <: \tau'_{\phi(i)}$.

Proof.

By induction on the size of the derivation, by case on the last applied rule.

- (Sub-Reflexivity). Take ϕ be the identity of $[1..n]$. Immediate.
- (Sub-Transitivity). There exists τ' such that $\tau' <: \{(\ell'_j : \tau'_j)_{j\in[1..m]}\}$ (**1**) and $\tau <: \tau'$ (**2**). We first apply the induction hypothesis on (1) and we get $\tau' = \{(\ell''_i : \tau''_i)_{i\in[1..k]}\}$ and there exists an injective function ϕ_1 from $[1..k]$ to $[1..m]$ such that forall $i \in [1..k]$, $\ell''_i = \ell'_{\phi_1(i)}$ and $\tau''_i <: \tau'_{\phi_1(i)}$. Applying the induction hypothesis a second time on (2) gives us $\tau = \{(\ell_i : \tau_i)_{i\in[1..n]}\}$ and there exists an injective function ϕ_2 from $[1..n]$ to $[1..k]$ such that forall $i \in [1..k]$, $\ell_i = \ell''_{\phi_2(i)}$ and $\tau_i <: \tau''_{\phi_2(i)}$. Take $\phi = \phi_1 \circ \phi_2$. It is injective since composing two injective functions leads to an injective function. We have $\ell_i = \ell''_{\phi_2(i)} = \ell'_{\phi_1(\phi_2(i))}$ and $\tau_i <: \tau''_{\phi_2(i)} <: \tau'_{\phi_2(\phi_1(i))}$.

□

Inversion Lemmas about the subtyping relation

Lemma (Inversion of the subtyping relation between records)

If $\tau <: \{(\ell'_j : \tau'_j)_{j\in[1..m]}\}$ holds, then $\tau = \{(\ell_i : \tau_i)_{i\in[1..n]}\}$ and there exists an injective function ϕ from $[1..n]$ to $[1..m]$ such that forall $i \in [1..n]$, $\ell_i = \ell'_{\phi(i)}$ and $\tau_i <: \tau'_{\phi(i)}$.

Proof.

(Continuation)

- ▷ The last rule is (Sub-Record-Depth). Take ϕ to be the identity. The conclusion is in the hypothesis of the rule.
- ▷ The last rule is (Sub-Record-Permutation). Take $\phi = \sigma$ and conclude by the reflexivity of the subtyping relation.
- ▷ The last rule is (Sub-Record-Width). The ϕ to be the identity again and conclude by the reflexivity of the subtyping relation.

□

Inversion Lemmas about the typing judgment

Lemma (Inversion of abstraction typing derivations)

If $\Gamma \vdash \lambda(x : \tau).t : \tau_1 \rightarrow \tau_2$ holds then $\tau_1 <: \tau$ and $\Gamma, (x : \tau) \vdash t : \tau_2$.

Proof.

By induction over typing derivations and by case on the last applied rule.

- The last rule is (Abs). In that case $\tau_1 = \tau$ and the hypothesis of the rule is the conclusion of the lemma.
- The last rule is (Sub). By hypothesis, there exists τ' such that $\tau' <: \tau_1 \rightarrow \tau_2$ and $\Gamma \vdash \lambda(x : \tau).t : \tau'$ (**1**). By Lemma **??**, there exists τ_1' and τ_2' such that $\tau' = \tau_1' \rightarrow \tau_2'$ with $\tau_1 <: \tau_1'$ (**2**) and $\tau_2' <: \tau_2$(**3**). Therefore, (1) is $\Gamma \vdash \lambda(x : \tau).t : \tau_1' \rightarrow \tau_2'$. By induction hypothesis applied to this judgment, we get that $\tau_1' <: \tau$ (**4**) and $\Gamma, (x : \tau) \vdash t : \tau_2'$(**5**). (4) and (2) gives $\tau_1 <: \tau$. Rule (Subsomption) applied to (5) and (3), gives $\Gamma, (x : \tau) \vdash t : \tau_2$.

□

Inversion Lemmas about the typing judgment

Lemma (Inversion of record typing derivations)

If $\Gamma \vdash \{(\ell_i = t_i)_{i \in [1..n]}\} : \{(\ell'_j : \tau_j)_{j \in [1..m]}\}$ holds then there exists an injective function ϕ from $[1..n]$ to $[1..m]$ such that for all $i \in [1..n]$, $\ell_i = \ell'_{\phi(i)}$ and $\Gamma \vdash t_i : \tau_{\phi(i)}$.

Proof.

By induction over typing derivations and by cases on the last applied rule.

- The last rule is (Record). Hence $n = m, \forall i, \ell_i = \ell'_i$ and $\Gamma \vdash t_i : \tau_i$. ϕ is the identity.
- The last rule is (Subsomption). By hypothesis of this rule, $\Gamma \vdash \{(\ell_i = t_i)_{i \in [1..n]}\} : \tau$ (**1**) and $\tau <: \{(\ell'_j : \tau_j)_{j \in [1..m]}\}$ (**2**). By Lemma (lemma:inverse-sub-records??), $\tau = \{(\ell''_i : \tau'_i)_{i \in [1..k]}\}$ and there exists ϕ_1 injective such that $\ell''_i = \ell'_{\phi_1(i)}$ and $\tau'_i <: \tau_{\phi_1(i)}$. Hence, (1) rewrites into $\Gamma \vdash \{(\ell_i = t_i)_{i \in [1..n]}\} : \{(\ell''_i : \tau'_i)_{i \in [1..k]}\}$ (**3**). Applying the induction hypothesis to this judgment leads to the existence of ϕ_2, an injective function from $[1..n]$ to $[1..k]$ such that for all $i \in [1..n]$, $\ell_i = \ell''_{\phi_2(i)}$ and $\Gamma \vdash t_i : \tau'_{\phi_2(i)}$ (**4**). Hence $\ell_i = \ell'_{\phi_1(\phi_2(i))}$. Besides, applying (Subsomption) on these judgments using $\forall i \in [1..n]$, $\tau'_{\phi_2(i)} <: \tau_{\phi_1(\phi_2(i))}$ gives $\forall i \in [1..n]$, $\Gamma \vdash t_i : \tau_{\phi_1(\phi_2(i))}$ (**5**). So we can take $\phi = \phi_1 \circ \phi_2$.

□

Subtitution Lemma

Lemma

Typing is stable by substitution If $\Gamma(x : \tau_1) \vdash t : \tau$ and $\Gamma \vdash u : \tau_1$, then $\Gamma \vdash t[x \mapsto u] : \tau$

Proof.

By induction over typing derivations and by cases on the the last applied rule of the first typing derivation. The cases for (Abs), (App) and (Var) are the same as the proof in STLC.

- ▷ The last rule is (Sub). We have $\Gamma(x : \tau_1) \vdash t : \tau'$ and $\tau' <: \tau$ (**1**). The induction hypothesis gives $\Gamma \vdash t[x \mapsto u] : \tau'$. Applying (Subsomption) on this judgment with (1) gives the conclusion.
- ▷ The last rule is (Record) or (Field). Immediate by induction hypothesis.

□

Subtyping relation on records

Lemma

Let n, m be two integers greater than 1. If ϕ is an injective function from $[1..n]$ to $[1..m]$ and $\ell_i = \ell'_{\phi(i)}$ then $\{(\ell_i : \tau_{\phi(i)})_{i \in [1..n]}\} <: \{(\ell'_j : \tau_j)_{j \in [1..m]}\}$ holds.

Proof.

By induction over $m - n$. □

Subject reduction

Lemma (Preservation of types through reduction)

If $\Gamma \vdash t : \tau$ and $t \rightarrow t'$ then $\Gamma \vdash t' : \tau$.

Proof.

By induction over typing derivations. Notice that the rules (Abs) and (Var) are absurd since they assume that t is a value and that it reduces to t'.

- The last rule is (App). The same reasoning as for STLC applies: the hypotheses of this rule are $\Gamma \vdash u : \tau_1 \rightarrow \tau_2$ ($\mathbf{1}$) and $\Gamma \vdash w : \tau_1$ ($\mathbf{2}$) where $t = u\, w$.
  We reason by case on the last reduction rule used to derive $t \rightarrow t'$.
  - The last reduction is a toplevel β-reduction. Hence, $u = \lambda(x : \tau).a$, w is a value and $t' = a[x \mapsto w]$. By Lemma **??** applied to (1), we have that $\tau_1 <: \tau$ ($\mathbf{3}$) and $\Gamma, (x : \tau) \vdash a : \tau_2$. Applying (Subsomption) to (2) using (3) leads to $\Gamma \vdash w : \tau$. Now, by the Substitution Lemma, this implies $\Gamma \vdash a[x \mapsto w] : \tau_2$.
  - The last reduction is a reduction under a context C. We have $C[t] \rightarrow C[t']$ (**4**) where $t \rightarrow t'$ and $C = C_0\, w$ or $C = u\, C_0$ with u being a value. This gives rise to two subcases. If $C = C_0\, w$, then (4) implies that $u = C_0[t] \rightarrow C_0[t']$ (**5**). Hence, by the induction hypothesis applied to (1) and (5), we get that $\Gamma \vdash C_0[t'] : \tau_1 \rightarrow \tau_2$. By (App) on this judgment and (2), we have $\Gamma \vdash C_0[t']\, w : \tau_2$. If $C = u\, C_0$, then (4) implies that $w = C_0[t] \rightarrow C_0[t']$ (**6**). By induction hypothesis applied to (2) and (6), we get that $\Gamma \vdash C_0[t'] : \tau_2$. By (App) on this judgment and (3), we have $\Gamma \vdash u\, C_0[t'] : \tau_2$.

□

Subject reduction

Lemma (Preservation of types through reduction)

If $\Gamma \vdash t : \tau$ and $t \rightarrow t'$ then $\Gamma \vdash t' : \tau$.

Proof.

(Continued...)

- ▷ The last rule is (Sub). The hypotheses of this rule are $\Gamma \vdash t : \tau'$ (**1**) and $\tau' <: \tau$ (**2**). The induction hypothesis applied to (1) gives $\Gamma \vdash t' : \tau'$. The rule (Sub) on this judgment and (2) leads to the desired conclusion.
- ▷ The last rule is (Field). The hypothesis of this rule is $\Gamma \vdash u : \{(\ell'_j : \tau_j)_{j\in[1..m]}\}$ (**1**) with $t = u.\ell_k$. We now consider each possible case of reduction.
  - ▷ The last reduction rule is a reduction under context . If $u = \{(\ell_i = t_i)_{i\in[1..n]}\}$, then by Lemma **??** on (1), there exists an injective function ϕ from $[1..n]$ to $[1..m]$ such that for all $i \in [1..n]$, $\ell_i = \ell'_{\phi(i)}$ and $\Gamma \vdash t_i : \tau_{\phi(i)}$ (**2**). There exists some $l < n$ such that $\forall i < l, t_i$ is a value, $\forall i \geq l, t_i$ is not a value, and $t_l \rightarrow t'_l$. Applying the induction hypothesis on this judgment and (2) instantiated for $i = l$, we get that $\Gamma \vdash t'_l : \tau_{\phi(l)}$ (**3**). By (Record) on this judgement, (2) (except on l), and (3), $\Gamma \vdash C[t'_l] : \{(\ell_i : \tau_{\phi(i)})_{i\in[1..n]}\}$(**4**). Besides $\{(\ell_i : \tau_{\phi(i)})_{i\in[1..n]}\} <: \{(\ell'_j : \tau_j)_{j\in[1..m]}\}$(**5**), since ϕ is an injective function. By (Sub) on (5) and (4), $\Gamma \vdash C[t'_l] : \{(\ell'_j : \tau_j)_{j\in[1..m]}\}$. (Field) gives the conclusion. Other context shapes are treated similarly.

□

Subject reduction

Lemma (Preservation of types through reduction)

If $\Gamma \vdash t : \tau$ and $t \rightarrow t'$ then $\Gamma \vdash t' : \tau$.

Proof.

(Continued...)

- ▷ The last rule is (Field) (continued).
  The hypothesis of this rule is $\Gamma \vdash u : \{(\ell'_j : \tau_j)_{j \in [1..m]}\}$ **(1)** with $t = u.\ell_k$. We now consider each possible case of reduction.
  - ▷ The last reduction rule is a field projection . We have $u = \{(\ell_i = v_i)_{i \in [1..n]}\}$ and $t' = v_k$. Lemma **??** on (1), there exists an injective function ϕ from $[1..n]$ to $[1..m]$ such that for all $i \in [1..n]$, $\ell_i = \ell'_{\phi(i)}$ and $\Gamma \vdash t_i : \tau_{\phi(i)}$ **(2)**. Hence, $\Gamma \vdash v_k : \tau_{\phi(k)}$.
- ▷ The last rule is (Record). The hypotheses of this rule are $\Gamma \vdash t_i : \tau_i$**(1)** for all $i \leq n$ where $t = \{(\ell_i = t_i)_{i \in [1..n]}\}$. The only possible reduction rule is a reduction under context with a record context where for some k, $t_k \rightarrow t'_k$. Applying the induction hypothesis on this term t_k leads to $\Gamma \vdash t'_k : \tau_k$. Applying (Record) on this judgment and preserving (1) for other $i \neq k$ leads to the conclusion.

□

Canonical forms

Lemma (Canonical forms of functions)

If $\emptyset \vdash v : \tau_1 \rightarrow \tau_2$ then v is a λ-abstraction.

Proof.

By cases over the syntax of values. v cannot be a variable because it is closed. If v were a record or an integer, the type of v would not be an arrow. (Even in the case where the last typing rule of the derivation is a (Sub), only arrow types are subtypes of arrow types so the same argument applies on the hypothesis judgment.) □

Canonical forms

Lemma (Canonical forms of records)

If $\emptyset \vdash v : \{(\ell_i : \tau_i)_{i \in [1..n]}\}$ then v is a record of the shape $\{(\ell_j = t_j)_{j \in J \subset [1..n]}\}$.

Proof.

Same proof as for the previous lemma. □

Progress

Lemma (Well-typed terms which are not values are reducible)

If $\emptyset \vdash t : \tau$ then $t \rightarrow t'$ or t is a value.

Proof.

By induction over typing derivations.

- The last applied rule is (Var). Absurd since t is closed, so it cannot be a variable.
- The last applied rule is (Lam). t is a value.
- The last applied rule is (App). $t = t_1\ t_2$, so it is not a value and we must show that it reduces. The hypotheses of the typing rule are $\emptyset \vdash t_1 : \tau_1 \rightarrow \tau_2$ and $\emptyset \vdash t_2 : \tau_1$. By applying twice the induction hypothesis on these judgments, we get several subcases.
    - t_1 and t_2 are values. By the Lemma **??**, t_1 is a λ-abstraction. Hence, t is a β-redex.
    - t_1 is not a value but is reducible. Apply a reduction under context to t with $C = []t_2$.
    - t_1 is a value and t_2 is not a value but is reducible. Apply a reduction under context to t with $C = t_1\,[]$.
- The last applied rule is (Field). In that case, $t = r.\ell$. Hence, r has a record type. By induction hypothesis, r is a value or r is reducible.
    - r is a value. By Lemma **??**, it must be a record and hence, t can be reduced by a field projection.
    - r is reducible. Apply a reduction under context to t with $C = [].\ell$.

□

Progress

Lemma (Well-typed terms which are not values are reducible)

If $\emptyset \vdash t : \tau$ then $t \rightarrow t'$ or t is a value.

Proof.

(Continued...)

- The last applied rule is (Record). In that case, $t = \{(\ell_i = t_i)_{iin[1..n]}\}$.
  - For all i, t_i is a value. The term t is a value.
  - t_j is the smallest index such that t_j is not a value. Perform a reduction under context with $C = \{(t_i = v_i)_{i<j}; \ell_j = []; (\ell_k = t_k)_{k>j}\}$.
- The last applied rule is (Sub). The hypotheses of these rules are $\emptyset \vdash t : \tau'$ and $\tau' <: \tau$. The conclusion is a direct consequence of the induction hypothesis.

□

Plan

Deciding the subtyping relation

Question 3.1

Given two types τ_1 and τ_2, can you find an algorithm that decides if $\tau_1 <: \tau_2$ holds? Prove that it indeed terminates and that it is correct.

Answer

The algorithm $\mathsf{is}_{<:}(\tau_1, \tau_2)$ is defined by structural recursion over τ_1 and τ_2.

- If $\tau_2 = \top$, returns **true**.
- If $\tau_1 = \rho_1 \rightarrow \rho'_1$ and $\tau_2 = \rho_2 \rightarrow \rho'_2$, then returns $\mathsf{is}_{<:}(\rho_2, \rho_1) \wedge \mathsf{is}_{<:}(\rho'_1, \rho'_2)$.
- If $\tau_1 = \{(\ell_{1,i} : \tau_{1,i})_{i \in [1..n]}\}$ and $\tau_2 = \{(\ell_{2,j} : \tau_{2,j})_{j \in [1..m]}\}$, returns $\bigwedge_{i \in [1..n]}[(\exists j, \ell_{1,i} = \ell_{2,j} \wedge \mathsf{is}_{<:}(\tau_{1,i}, \tau_{2,j})]$.
- If $\tau_1 = \textbf{nat}$ and $\tau_2 = \textbf{nat}$ returns **true**.
- Returns **false** for all other cases.

Deciding the subtyping relation

The termination of the algorithm is immediate since the recursion is always performed on sub-terms of the input types.

The correctness of the algorithm is stated as follows:

Lemma

$\tau_1 <: \tau_2$ holds if and only if $\text{is}_{<:}(\tau_1, \tau_2)$ returns **true**.

Proof.

We first prove that $\tau_1 <: \tau_2$ holds implies $\text{is}_{<:}(\tau_1, \tau_2)$ returns **true** by induction over $\tau_1 <: \tau_2$. All cases are immediate by induction hypothesis. The other direction is also immediate by induction over pair of types. □

Observing the interaction between (Sub) and (Abs)

Question 3.2

Consider a typing derivation whose conclusion is obtained by an application of the rule (Abs) immediately preceded by an application of the (Subsomption) rule. Find a typing derivation for the same judgment but where (Subsomption) rule has been pushed down in the typing derivation, i.e. find a way to commute (Abs) and (Subsomption) in the previous typing derivation.

Observing the interaction between (Sub) and (Abs)

Answer

Let us consider the following typing derivation:

$$\text{(Abs)}\ \dfrac{\text{(Sub)}\ \dfrac{\Gamma; (x : \tau) \vdash t : \tau_2' \qquad \tau_2' <: \tau_2}{\Gamma; (x : \tau) \vdash t : \tau_2}}{\Gamma \vdash \lambda(x : \tau).t : \tau \to \tau_2}$$

We can push the (Sub) rule downside to get the same conclusion:

$$\text{(Sub)}\ \dfrac{\text{(Abs)}\ \dfrac{\Gamma; (x : \tau) \vdash t : \tau_2'}{\Gamma \vdash \lambda(x : \tau).t : \tau \to \tau_2'} \qquad \dfrac{\tau <: \tau \qquad \tau_2' <: \tau_2}{\tau \to \tau_2' <: \tau \to \tau_2}}{\Gamma \vdash \lambda(x : \tau).t : \tau \to \tau_2}$$

Observing the interaction between (Sub) and (App)

Question 3.3

Consider now the case for a typing derivation whose conclusion is obtained by an application of rule (App) immediately preceded by an application of the (Subsomption) rule on the left-hand-side hypothesis. Is it possible to find a typing derivation where the (Subsomption) is pushed at the bottom? Same question if the (Subsomption) rule is located on the right-hand side.

Observing the interaction between (Sub) and (App)

Answer

To start with, let us consider now a typing derivation where (Sub) is applied on the left-hand-side of the application:

$$\text{(App)}\ \dfrac{\text{(Sub)}\ \dfrac{\Gamma \vdash t : \rho_1 \to \rho_2 \qquad \dfrac{\tau_1 <: \rho_1 \qquad \rho_2 <: \tau_2}{\rho_1 \to \rho_2 <: \tau_1 \to \tau_2}}{\Gamma \vdash t : \tau_1 \to \tau_2} \qquad \Gamma \vdash u : \tau_1}{\Gamma \vdash t\,u : \tau_2}$$

In that case, pushing (Sub) downside can be done only if we insert a new (Sub) to convert the type of u:

$$\text{(Sub)}\ \dfrac{\text{(App)}\ \dfrac{\Gamma \vdash t : \rho_1 \to \rho_2 \qquad \text{(Sub)}\ \dfrac{\Gamma \vdash u : \tau_1 \qquad \tau_1 <: \rho_1}{\Gamma \vdash u : \rho_1}}{\Gamma \vdash t\,u : \rho_2} \qquad \rho_2 <: \tau_2}{\Gamma \vdash t\,u : \tau_2}$$

Observing the interaction between (Sub) and (App)

Answer

Consider now a typing derivation where (Sub) is applied on the right-hand-side of the application:

$$\text{(App)}\ \dfrac{\Gamma \vdash t : \tau_1 \to \tau_2 \qquad \dfrac{\Gamma \vdash u : \rho_1 \qquad \rho_1 <: \tau_1}{\Gamma \vdash u : \tau_1}\ \text{(Sub)}}{\Gamma \vdash t\, u : \tau_2}$$

Again, we cannot push the rule (Sub) down the derivation tree without inserting a (Sub) on the left-hand-side of the application.

Normalization of typing derivations

Question 3.5

By analogy with the proof of equivalence between the declarative and algorithmic type system of ML, introduce two explicitly-typed languages $e\lambda_{<:}$ and $x\lambda_{<:}$ for $\lambda_{<:}$ whose type erasures match $\lambda_{<:}$. The language $e\lambda_{<:}$ is equipped with the same rules as the declarative type system of $\lambda_{<:}$. The language $x\lambda_{<:}$ is equipped with algorithmic typing rules. Define a normalization procedure from the typing derivations of $e\lambda_{<:}$ to the typing derivations of $x\lambda_{<:}$ whose judgment is written:

$$\Gamma \vdash t \in e\lambda_{<:} : \tau \Rightarrow t' \in x\lambda_{<:} : \tau'$$

Normalization of typing derivations

The first explicitly typed language $e\lambda_{<:}$ embeds a syntactic construction in terms to make explicit the application of subsomption rules:

$$t \in e\lambda_{<:} ::= x \mid \lambda(x : \tau).t \mid t\ t \mid \{(\ell_i = t)_{i \in [1..n]}\} \mid t.\ell \mid (\mathbf{t} : \tau_1 <: \tau_2)$$

From a typing derivation $\Gamma \vdash t : \tau$ in $\lambda_{<:}$, one can easily construct $\Gamma \vdash t \in e\lambda_{<:} : \tau$ since the typing derivations of $\lambda_{<:}$ and the terms of $e\lambda_{<:}$ are isomorphic.

The difficult part is to show how any typing derivation can be normalized to the terms of the following syntax:

$$t \in x\lambda_{<:} ::= x \mid \lambda(x : \tau).t \mid \mathbf{t}\ (\mathbf{t} : \tau_1 <: \tau_2) \mid \{(\ell_i = t)_{i \in [1..n]}\} \mid t.\ell$$

Normalization of typing derivations

Let us start with the “obvious” rules:

Var

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x \in e\lambda_{<:} : \tau \Rightarrow x \in x\lambda_{<:} : \tau}$$

Abs

$$\frac{\Gamma; (x : \tau) \vdash t \in e\lambda_{<:} : \tau_2 \Rightarrow t' \in x\lambda_{<:} : \tau_2'}{\Gamma \vdash \lambda(x : \tau).t \in e\lambda_{<:} : \tau \rightarrow \tau_2 \Rightarrow \lambda(x : \tau).t' \in x\lambda_{<:} : \tau \rightarrow \tau_2'}$$

Field

$$\frac{\begin{array}{c}\Gamma \vdash t \in e\lambda_{<:} : \{(\ell_i : \tau_i)_{i \in [1..n]}\} \Rightarrow t' \in x\lambda_{<:} : \{(\ell_i' : \tau_i')_{i \in [1..m]}\} \\ \ell = \ell_k \qquad \ell = \ell_l'\end{array}}{\Gamma \vdash t.\ell \in e\lambda_{<:} : \tau_k \Rightarrow t'.\ell \in x\lambda_{<:} : \tau_l'}$$

Record

$$\frac{\forall i \qquad t_i \in e\lambda_{<:} : \tau_i \Rightarrow t_i' \in x\lambda_{<:} : \tau_i'}{\begin{array}{c}\Gamma \vdash \{(\ell_i = t_i)_{i \in [1..n]}\} \in e\lambda_{<:} : \{(\ell_i : \tau_i)_{i \in [1..n]}\} \\ \Rightarrow \{(\ell_i = t_i')_{i \in [1..n]}\} \in x\lambda_{<:} : \{(\ell_i : \tau_i')_{i \in [1..n]}\}\end{array}}$$

Normalization of typing derivations

As coined before, a coercion must be written on the right-hand side of each application.

App

$$\frac{\begin{array}{c}\Gamma \vdash t \in e\lambda_{<:} : \rho_1 \to \rho_2 \Rightarrow t' \in x\lambda_{<:} : \tau_1 \to \tau_2 \\ \Gamma \vdash u \in e\lambda_{<:} : \tau_1 \Rightarrow u' \in x\lambda_{<:} : \rho\end{array}}{\Gamma \vdash t\ u \in e\lambda_{<:} : \rho_2 \qquad \Rightarrow t'(u' : \rho <: \tau_1) \in x\lambda_{<:} : \tau_2}$$

Coercion

$$\frac{\Gamma \vdash t \in e\lambda_{<:} : \tau_1 \Rightarrow t' \in x\lambda_{<:} : \tau_2'}{\Gamma \vdash (t : \tau_1 <: \tau_2) \in e\lambda_{<:} : \tau_2 \Rightarrow t' \in x\lambda_{<:} : \tau_2'}$$

One can show that if $\Gamma \vdash t \in e\lambda_{<:} : \tau \Rightarrow t' \in x\lambda_{<:} : \tau'$ holds then $\Gamma \vdash t' \in x\lambda_{<:} : \tau'$ and $\tau' <: \tau$ hold as well.

Plan

Plan

Reference types

Question 4.1

An extension of $\lambda_{<:}$ with references (mutable cells) requires the introduction of a type $\textbf{ref}\ \tau$. How would you write the subtyping rule for these reference types? Justify your answer.

Answer

Reference types are observationally equivalent to the following (recursive) record type:

$$
\begin{array}{l}
\textbf{ref}\ \tau = \{ \\
\quad \text{write} : \tau \times \textbf{ref}\ \tau \to \textbf{unit}; \\
\quad \text{read} : \textbf{ref}\ \tau \to \tau \\
\}
\end{array}
$$

Applying the subtyping rules to $\text{ref}\ \tau <: \text{ref}\ \tau'$ leads to $\tau <: \tau'$ (since τ appears in covariant position in the read field) and $\tau' <: \tau$ (since τ appears in contravariant position in the write field). Therefore **ref** must be **invariant**:

$$
\frac{\tau <: \tau' \qquad \tau' <: \tau}{\textbf{ref}\ \tau <: \textbf{ref}\ \tau'}
$$

Plan

Bottom

Question 4.2

Imagine that we introduce $\bot$, a dual to $\top$, which is a subtype of any type, i.e. $\bot <: \tau$. Why must this type be empty? (i.e. It cannot be inhabited to preserve type satefy.) What would be its usage, then?

Answer

Let us assume for a second that $v : \bot$, then $v :$ **nat** and $v : \top \rightarrow \top$ so v should be both an integer and a λ-abstraction. Hence there is a contradiction.

Such a type would nonetheless be interesting to denote unreachable control-flow point, aka dead code.