CS 456

Programming Languages
Fall 2024

Week 7
Subtyping

Inclusion Polymorphism

- Polymorphism = allowing a single definition to be used with different types in different contexts.
  - Parametric polymorphism: behavior is the same for all instantiations (System F)
  - Ad-hoc polymorphism: behavior is based on the type of its arguments (overloading)

- One of the defining characteristics of Object-Oriented languages is their support for inclusion polymorphism
  - Inclusion polymorphism allows for code that needs *at least* a T in a context

A Calculus for Subtyping

- ★ Begin with a simple language for studying subtyping
- ★ Key extension is Records (labeled products)

t ::= x | λx:T.t | t t | n ∈ ℕ | i ∈ ℝ | t + t | …

| ⟨l1=t1, …, ln = tn⟩ ⇐ Records

| t.l ⇐ Projection

v ::= λx:T.t | n ∈ ℕ | i ∈ ℝ | ⟨l1=v1, …, ln = vn⟩

can be empty < >

T ::= T → T | ℝ | ℕ

| ⟨l1:T1, …, ln : Tn⟩ ⇐ Record Types

| Top

A Calculus for Subtyping

- Begin with a simple language for studying subtyping

- Key extension is Records (labeled products)

t ::= x | λx:T.t | t t | … | ⟨l1=t1, …, ln = tn⟩ ⇐ Records

| t.l ⇐ Projection

T ::= T → T | ℝ | … | ⟨l1:T1, …, ln : Tn⟩ ⇐ Record Types

Point ≡ ⟨x :ℝ, y :ℝ⟩

Dist ≡ λp:Point.$\sqrt{p.x^2 + p.y^2}$

Dist ⟨x : 2, y : 2⟩ ⟶* 2.82…

ColorPoint ≡ ⟨x :ℝ, y :ℝ, R:ℕ, G:ℕ, B:ℕ⟩

Subsumption

★Would like this to typecheck:

Dist ⟨x=2, y=2, R=0, G=140, B=255⟩

★Let's extend existing type system!

★Key Idea: the subsumption rule:

$$\frac{\Gamma \vdash t_1 : T_1 \quad T_1 <: T_2}{\Gamma \vdash t_1 : T_2}\ \textsc{Tsub}$$

$$\frac{\vdash \text{Dist} : \text{Point} \to \mathbb{R} \qquad \dfrac{\dfrac{\Gamma \vdash \langle x=2, y=2, R=0, G=140, B=255\rangle : \text{ColorPoint} \qquad \text{ColorPoint} <: \text{Point}}{\Gamma \vdash \langle x=2, y=2, R=0, G=140, B=255\rangle : \text{Point}}\ \textsc{Tsub}}{\vdash \text{Dist}\ \langle x=2, y=2, R=0, G=140, B=255\rangle : \mathbb{R}}}{}$$

Subsumption

Would like this to typecheck:

Dist ⟨x=2, y=2, R=0, G=140, B=255⟩

$$\frac{\Gamma \vdash t_1 : T_1 \quad T_1 <: T_2}{\Gamma \vdash t_1 : T_2}\ \text{TSUB}$$

How to define T1 <:T2?

Substitutability: If T1 <:T2, then any value of type T1 must be usable in every way a T2 is.

The difficulty is ensuring this is safe (i.e. doesn't break type safety)!

Subtype Relation

What key properties should a substitutability relation satisfy?

- Should be transitive:

$$\frac{S <: T \quad T <: U}{S <: U}\ \textsc{STrans}$$

- And reflexive (not strictly necessary, but makes theory nicer)

$$\frac{}{S <: S}\ \textsc{SRefl}$$

- How to relate concrete types?

$$\frac{}{\mathbb{N} <: \mathbb{R}}\ \textsc{SNum}$$

Subtyping Records

What sorts of pairs of record types satisfy substitutability?

- A subtype should be able to add fields at the end:

$$\langle x : \mathbb{R}, y : \mathbb{R}, R : \mathbb{N} \rangle <: \langle x : \mathbb{R}, y : \mathbb{R} \rangle$$

$$\frac{}{\langle l_1 : T_1, \ldots, l_n : T_n, \ldots, l_{n+k} : T_{n+k} \rangle <: \langle l_1 : T_1, \ldots, l_n : T_n \rangle} \text{ SBWIDTH}$$

- A subtype should be able to reorder fields:

$$\langle y : \mathbb{R}, x : \mathbb{R} \rangle <: \langle x : \mathbb{R}, y : \mathbb{R} \rangle$$

$$\frac{\langle k_1 : T_1, \ldots, k_n : T_n \rangle \text{ is a permutation of } \langle l_1 : T_1, \ldots, l_n : T_n \rangle}{\langle l_1 : T_1, \ldots, l_n : T_n \rangle <: \langle k_1 : T_1, \ldots, k_n : T_n \rangle} \text{ SBPERM}$$

Subtyping Records

What sorts of pairs of record types satisfy substitutability?

A subtype should be able to add fields at the end:

$$\frac{}{\langle l_1{:}T_1,\ \ldots,\ l_n{:}T_n,\ \ldots,\ l_{n+k}{:}T_{n+k}\rangle <: \langle l_1{:}T_1,\ \ldots,\ l_n{:}T_n\rangle}\ \textsc{SBWidth}$$

A subtype should be able to reorder fields:

$$\frac{\langle k_1{:}T_1,\ \ldots,\ k_n{:}T_n\rangle \text{ is a permutation of } \langle l_1{:}T_1,\ \ldots,\ l_n{:}T_n\rangle}{\langle l_1{:}T_1,\ \ldots,\ l_n{:}T_n\rangle <: \langle k_1{:}T_1,\ \ldots,\ k_n{:}T_n\rangle}\ \textsc{SBPerm}$$

A subtype should be able to specialize field types:

$$\langle x : \mathbb{R}, y : \mathbb{R}, R{:}\mathbb{R}\rangle <: \langle x : \mathbb{R}, y : \mathbb{R}, R{:}\mathbb{N}\rangle$$

$$\frac{\dfrac{}{T_i <: S_i}}{\langle l_1{:}T_1,\ \ldots,\ l_n{:}T_n\rangle <: \langle l_1{:}S_1,\ \ldots,\ l_n{:}S_n\rangle}\ \textsc{SBDepth}$$

Subtyping Functions

★ Subtyping helps with our function application problem!

$$\vdash \text{Dist}\ \langle x{=}2,\ y{=}2,\ R{=}0,\ G{=}140,\ B{=}255\rangle : \mathbb{R}\ \textsc{SubWidth}$$

$$\dfrac{\Gamma \vdash \langle x{=}2,\ y{=}2,\ R{=}0,\ G{=}140,\ B{=}255\rangle : \text{ColorPoint} \quad \text{ColorPoint} <: \text{Point}}{\Gamma \vdash \langle x{=}2,\ y{=}2,\ R{=}0,\ G{=}140,\ B{=}255\rangle : \text{Point}}\ \textsc{TSub}$$

Subtyping Functions

- Subtyping helps with our function application problem* …
- What about this expression:

DistMoved ≡ λt:Point → Point. p:Point.
lDist (t p) - Dist p l

Flip ≡ λp:Point. ⟨x = -(p.x), y = -(p.y)⟩

⊢ DistMoved : (Point → Point) → Point → ℝ

⊢ DistMoved Flip ⟨x=2, y=2⟩ : ℝ

DistMoved Flip ⟨x=2, y=2⟩ ⟶* 0

Subtyping Functions

- Subtyping helps with our function application problem* …
- What about this expression:

$$\mathsf{DistMoved} \equiv \lambda t{:}\mathsf{Point} \to \mathsf{Point}.\ p{:}\mathsf{Point}.\ \mathsf{lDist}\ (t\ p) - \mathsf{Dist}\ p\ \mathsf{l}$$

$$\mathsf{FlipGreen} \equiv \lambda p{:}\mathsf{Point}.\ \langle x{=}{-}(p.x), y{=}{-}(p.y), R{=}0, G{=}255, B{=}0\rangle$$

$$\vdash \mathsf{DistMoved} : (\mathsf{Point} \to \mathsf{Point}) \to \mathsf{Point} \to \mathbb{R}$$

$$\vdash \mathsf{FlipGreen} : (\mathsf{Point} \to \mathsf{ColorPoint})$$

$$\nvdash \mathsf{DistMoved}\ \mathsf{FlipGreen}\ \langle x{=}2, y{=}2\rangle$$

$$\mathsf{DistMoved}\ \mathsf{FlipGreen}\ \langle x{=}2, y{=}2\rangle \longrightarrow^{*} 0$$

Subtyping Functions

- Subtyping helps with our function application problem* …
- What about this expression:

```
DistMoved ≡ λt:Point → Point. p:Point.
                lDist (t p) - Dist p l
FlipGreen ≡ λp:Point.
          ⟨x=-(p.x),y=-(p.y), R=0, G=255, B=0⟩
```

$$\frac{S_2 <: T_2}{T_1 \to S_2 <: T_1 \to T_2}\ \text{SB-ARROW}_2$$

DistMoved FlipGreen ⟨x=2, y=2⟩

$\vdash$ DistMoved FlipGreen ⟨x=2, y=2⟩ : $\mathbb{R}$

Subtyping Functions

- Subtyping helps with our function application problem* …
- What about this expression:

```
DistMoved ≡ λt:Point → Point. p:Point.
                |Dist (t p) - Dist p |
FlipGreener ≡ λp:ColorPoint.
        ⟨x=-(p.x),y=-(p.y), R=p.R, G=255, B=p.B⟩
```

DistMoved FlipGreener ⟨x=2, y=2⟩

Subtyping Functions

- Subtyping helps with our function application problem*…
- What about this expression:

DistMoved ≡ λt:Point → Point. p:Point.
  lDist (t p) - Dist p l

FlipGreener ≡ λp:ColorPoint.
  ⟨x=-(p.x),y=-(p.y), R=p.R, G=255, B=p.B⟩

$$\frac{T_1 <: S_1 \quad S_2 <: T_2}{S_1 \to S_2 <: T_1 \to T_2}$$ SB-ARROW

✗ DistMoved FlipGreener ⟨x=2, y=2⟩

Subtyping Functions

- Subtyping helps with our function application problem*!
- What about this expression:

DistMovedC ≡ λt:ColorPoint → Point. p:ColorPoint.
  |Dist (t p) - Dist p |

FlipGreener ≡ λp:ColorPoint.
  ⟨x=-(p.x),y=-(p.y), R=p.R, G=255, B=p.B⟩

$$\frac{T_1 <: S_1 \quad S_2 <: T_2}{S_1 \to S_2 <: T_1 \to T_2}$$ SB-ARROW

DistMovedC Flip ⟨x=2, y=2,R=0, G=0, B=0⟩

DistMovedC FlipGreen ⟨x=2, y=2,R=0, G=0, B=0⟩

DistMovedC FlipGreener ⟨x=2, y=2,R=0, G=0, B=0⟩

Variance

Variance is a property on the arguments of type constructors like function types $(A \to B)$, tuples $(A \times B)$, and record types

$F(A)$ is **covariant** over A if $A <: A'$ implies that $F(A) <: F(A')$

$F(B)$ is **contravariant** over B if $B' <: B$ implies that $F(B) <: F(B')$

$F(T)$ is **invariant** over T otherwise

$$\frac{S_1 <: T_1 \quad S_2 <: T_2}{S_1 \times S_2 <: T_1 \times T_2} \text{ SB-TUPLE}$$

$$\frac{T_1 <: S_1 \quad S_2 <: T_2}{S_1 \to S_2 <: T_1 \to T_2} \text{ SB-ARROW}$$

Classes Vs. Types

- A class provides implementations of an object's behavior
  Subclassing inherits behavior and changes it via extension and overriding
- A class also defines an interface (type) for its object's type
  A subtype is substitutable in terms of its field/method types

Inheritance Vs. Subtyping

- Consider three datatypes:
  Queues: FIFO insertion and deletion
  Stacks: LIFO insertion and deletion
  Dequeues: insert/delete from rear/front
- A dequeue implementation provides both a stack and a queue:
  Stack: only expose insert_front and delete_front operations
  Queue: only expose insert_rear and delete_front operations

Neither a Stack nor a Queue is a Dequeue though!

Inheritance Vs. Subtyping

- These are separate concepts!
  - Java confuses them by making subclasses be subtypes
  - Inheritance is a mechanism for code reuse
  - Subtyping is a mechanism for checking substitutability