Subtyping

Chapter 15

Benjamin Pierce

Types and Programming Languages

Varieties of Polymorphism

- Parametric polymorphism A single piece of code is typed generically
  - Imperative or first-class polymorphism
  - ML-style or let-polymorphism
- Ad-hoc polymorphism The same expression exhibit different behaviors when viewed in different types
  - Overloading
  - Multi-method dispatch
  - Intentional polymorphism
- Subtype polymorphism A single term may have many types using the rule of subsumption allowing to selectively forget information

Simple Typed Lambda Calculus

t ::=	terms
x	variable
λ x: T. t	abstraction
t t	application

T::=	types
$T \rightarrow T$	types of functions

Type Rules

$\Gamma \vdash t : T$

t ::= terms

x variable

λ x: T. t abstraction

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \text{ (T-VAR)}$$

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1 . t_2 : T_1 \to T_2} \text{ (T-ABS)}$$

T::= types

$T \to T$ types of functions

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \quad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \text{ (T-APP)}$$

Γ::= context

$\varnothing$ empty context

$\Gamma, x : T$ term variable binding

Records

New syntactic forms

t ::= ….		Terms:
	$\{l_i=t_i \ ^{i \in 1..n}\}$	record
	t.l	projection
v ::= ….		Values:
	$\{l_i=v_i \ ^{i \in 1..n}\}$	records
T ::= ….		types:
$\{l_i:T_i \ ^{i \in 1..n}\}$		record type

New Evaluation Rules extends $\rightarrow$

$\{l_i=v_i \ ^{i \in 1..n}\}.\ l_j \rightarrow v_j$ (E-ProjRCD)

$$\frac{t \rightarrow t'}{t.l \rightarrow t'.l} \quad \text{(E-Proj)}$$

$$\frac{t_j \rightarrow t'_j}{\{l_i=v_i \ ^{i \in 1..j-1},\ l_i=t_i \ ^{i \in j..n}\} \rightarrow \{l_i=v_i \ ^{i \in 1..j-1}, l_j=t'_j, l_k=t_k \ ^{k \in j+1..n}\}} \quad \text{(E-Tuple)}$$

New typing rules

$$\frac{\text{For each } i\ \Gamma \vdash t_i : T_i}{\Gamma \vdash \{l_i=t_i \ ^{i \in 1..n}\} : \{l_i: T_i \ ^{i \in 1..n}\}} \quad \text{(T-Tuple)}$$

$$\frac{\Gamma \vdash t: \{l_i: T_i \ ^{i \in 1..n}\}}{\Gamma \vdash t.l_j : T_j} \quad \text{(T-Proj)}$$

Record Example

$(\lambda r : \{x: \text{Nat}\}.\ r.x)\ \{x=0,\ y=0\ \}$

$\{x: \text{Nat}, y: \text{Nat}\} <: \{x: \text{Nat}\}$

$$\frac{\Gamma \vdash t : S \qquad S <: T}{\Gamma \vdash t : T} \text{ (T-SUB)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \quad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \text{ (T-APP)}$$

Healthiness of Subtypes

$$S <: S \qquad \text{(S-REFL)}$$

$$\frac{S <: U \qquad U <: T}{S <: T} \qquad \text{(S-TRANS)}$$

Width Record Subtyping

$\{l_i:T_i \; ^{i \in 1..n+k}\} <: \{l_i:T_i \; ^{i \in 1..n}\}$ (S-RCDWIDTH)

{x: Nat} has at least the field x of type Nat

{x: Nat, y : Nat} has at least the field x of type Nat and the field y of type Nat

{x: Nat, y: Bool} has at least the field x of type Nat and the field y of type Bool

{x: Nat, y: Nat} <: {x: Nat}

{x: Nat, y: Bool} <: {x: Nat}

Record Depth Subtyping

$$\frac{\text{For each i } S_i <: T_i}{\{l_i : S_i \ ^{i \in 1..n}\} <: \{l_i : T_i \ ^{i \in 1..n}\}} \text{ (S-RCDEPTH)}$$

{x: {a: Nat, b: Nat}, y: {m:Nat}} <: {x: {a: Nat}, y: {}}

{a:Nat, b: Nat} <: {a: Nat} (S-RCDWIDTH) {m: Nat} <: {} (S-RCDWIDTH)

$$\frac{}{\{x: \{a: Nat, b: Nat\}, y: \{m: Nat\}\} <: \{x: \{a: Nat\}, y: \{\}\}}\ (S\text{-}RCDEPTH)$$

{x: {a: Nat, b: Nat}, y: Nat} <: {x: {a: Nat}, y: Nat}}

{a:Nat, b: Nat} <: {a: Nat} (S-RCDWIDTH) Nat <: Nat (S-REFL)

$$\frac{}{\{x: \{a: \text{Nat}, b: \text{Nat}\}, y: \text{Nat}\} <: \{x: \{a: \text{Nat}\}, y: \{\}\}}\ (\text{S-RCDEPTH})$$

{x: {a: Nat, b: Nat}, y: {m: Nat}} <: {x: {a: Nat}}

{x: {a:Nat, b: Nat}}, {y: {m: Nat}} <: {x: {a: Nat}, b:Nat}} (S-RCDWIDTH)

{a: Nat, b: Nat} <: {a: Nat} (S-RCDWIDTH)

$$\frac{}{\{x: \{a: Nat, b: Nat\} <: \{x: \{a: Nat\}\}}\ (S\text{-}RCDEPTH)$$

$$\frac{}{\{x: \{a: Nat, b: Nat\}, y: \{m: Nat\}\} <: \{x: \{a: Nat\}\}}\ (S\text{-}TRANS)$$

Field Permutation

$$\frac{\{k_j{:}S_j\ ^{j \in 1..n}\} \text{ is a permutation of } \{l_i{:}T_i\ ^{i \in 1..n}\}}{\{k_j{:}S_j\ ^{j \in 1..n}\} <: \{l_i{:}T_i\ ^{i \in 1..n}\}} \quad \text{(S-RCDPERM)}$$

Record Subtying

- Forgetting fields (S-RCDWIDTH)
- Forgetting subrecords (S-RCDEPTH)
- Reordering fields (S-RCDPERM)

Naïve Handling of Functions

$$\frac{S_1 <: T_1 \qquad S_2 <: T_2}{S_1 \to S_2 <: T_1 \to T_2}\ \text{(S-ARROWN)}$$

$$\frac{n{:}\{x{:}\ \text{Nat}, y{:}\ \text{Nat}\} \vdash\ n.x + 1 : \text{Nat} \qquad n{:}\{x{:}\ \text{Nat}, y{:}\ \text{Nat}\} \vdash\ n.y + 2 : \text{Nat}}{n{:}\{x{:}\ \text{Nat}, y{:}\ \text{Nat}\} \vdash \{x{:}\ n.x + 1\ ,\ y{:}\ n.y + 2\} : \{x{:}\ \text{Nat}, y{:}\ \text{Nat}\}}\ \text{(T-RCD)}$$

$$\frac{}{\vdash \lambda n{:}\{x{:}\ \text{Nat}, y{:}\ \text{Nat}\}.\ \{x{:}\ n.x + 1\ ,\ y{:}\ n.y + 2\}) : \{x{:}\ \text{Nat}, y{:}\ \text{Nat}\} \to \{x{:}\ \text{Nat}, y{:}\ \text{Nat}\}}\ \text{(T-ABS)}$$

$\{x{:}\text{Nat}, y{:}\ \text{Nat}\} <:\ \{x{:}\ \text{Nat}\}$ (S-RCDWIDTH) $\quad \{x{:}\text{Nat}, y{:}\ \text{Nat}\} <:\ \{x{:}\ \text{Nat}, y{:}\ \text{Nat}\}$ (S-REFL)

$$\frac{}{\{x{:}\text{Nat}, y{:}\ \text{Nat}\} \to \{x{:}\text{Nat}, y{:}\ \text{Nat}\} <: \{x{:}\ \text{Nat}\} \to \{x{:}\ \text{Nat}, y{:}\ \text{Nat}\}}\ \text{(S-ARROWN)}$$

$$\frac{}{\vdash \lambda n{:}\{x{:}\ \text{Nat}, y{:}\ \text{Nat}\}.\ \{x{:}\ n.x + 1\ ,\ y{:}\ n.y + 2\}) : \{x{:}\ \text{Nat}\} \to \{x{:}\ \text{Nat}, y{:}\ \text{Nat}\}}\ \text{(T-SUB)}$$

$$\frac{\vdash 1 : \text{Nat}}{\vdash \{x{:}\ 1\} : \{x{:}\ \text{Nat}\}}\ \text{(T-RECD)}$$

$$\frac{}{\vdash \lambda n{:}\{x{:}\ \text{Nat}, y{:}\ \text{Nat}\}.\ \{x{:}\ n.x + 1\ ,\ y{:}\ n.y + 2\})\ \{x{:}1\}\ : \{x{:}\ \text{Nat}, y{:}\ \text{Nat}\}}\ \text{(T-APP)}$$

Handling of Functions

$$\frac{T_1 <: S_1 \qquad S_2 <: T_2}{S_1 \to S_2 <: T_1 \to T_2} \text{ (S-ARROW)}$$

- Arguments types are handled in reversed way (contravariant)
- Result types are handled in the same direction (covariant)

Top type

T <: Top (S-TOP)

Properties of the subtyping relation

- Preorder on types

SOS for Simple Typed Lambda Calculus

t ::=		terms
	x	variable
	λ x: T. t	abstraction
	t t	application

v::=		values
	λ x: T. t	abstraction values

$$t_1 \rightarrow t_2$$

$$\frac{t_1 \rightarrow t'_1}{t_1\ t_2 \rightarrow t'_1\ t_2} \quad \text{(E-APP1)}$$

$$\frac{t_2 \rightarrow t'_2}{v_1\ t_2 \rightarrow v_1\ t'_2} \quad \text{(E-APP2)}$$

$$(\lambda\, x{:}\, T_{11}.\ t_{12})\ v_2 \rightarrow [x \mapsto v_2]\ t_{12} \quad \text{(E-APPABS)}$$

T::=		types
	Top	maximum type
	T $\rightarrow$ T	types of functions

Type Rules

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \text{ (T-VAR)}$$

$$S <: S \quad \text{(S-REFL)}$$

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1.\ t_2 : T_1 \to T_2} \text{ (T-ABS)}$$

$$\frac{S <: U \qquad U <: T}{S <: T} \text{ (S-TRANS)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \text{ (T-APP)}$$

$$T <: \text{Top} \quad \text{(S-TOP)}$$

$$\frac{\Gamma \vdash t : S \qquad S <: T}{\Gamma \vdash t : T} \text{ (T-SUB)}$$

$$\frac{T_1 <: S_1 \qquad S_2 <: T_2}{S_1 \to S_2 <: T_1 \to T_2} \text{ (S-ARROW)}$$

Records

syntactic forms

Terms:
- $t ::= \ldots.$
- $\{l_i=t_i \ ^{i \in 1..n}\}$ record
- $t.l$ projection

Values:
- $v ::= \ldots.$
- $\{l_i=v_i \ ^{i \in 1..n}\}$ records

types:
- $T ::= \ldots.$
- $\{l_i:T_i \ ^{i \in 1..n}\}$ record type

Evaluation Rules extends $\rightarrow$

$\{l_i=v_i \ ^{i \in 1..n}\}.\ l_j \rightarrow v_j$ (E-ProjRCD)

$$\frac{t \rightarrow t'}{t.l \rightarrow t'.l} \quad \text{(E-Proj)}$$

$$\frac{t_j \rightarrow t'_j}{\{l_i=v_i \ ^{i \in 1..j-1},\ l_i=t_i \ ^{i \in j..n}\} \rightarrow \{l_i=v_i \ ^{i \in 1..j-1}, l_j=t'_j, l_k=t_k \ ^{k \in j+1..n}\}} \quad \text{(E-Tuple)}$$

New subtyping rules

$\{l_i:T_i \ ^{i \in 1..n+k}\} <: \{l_i:T_i \ ^{i \in 1..n}\}$ (S-RCDWIDTH)

typing rules

$$\frac{\text{For each } i\ \Gamma \vdash t_i : T_i}{\Gamma \vdash \{l_i=t_i \ ^{i \in 1..n}\} : \{l_i: T_i \ ^{i \in 1..n}\}} \quad \text{(T-Tuple)}$$

$$\frac{\text{For each } i\ S_i <: T_i}{\{l_i:S_i \ ^{i \in 1..n}\} <: \{l_i:T_i \ ^{i \in 1..n}\}} \quad \text{(S-RCDEPTH)}$$

$$\frac{\Gamma \vdash t: \{l_i: T_i \ ^{i \in 1..n}\}}{\Gamma \vdash t.l_j : T_j} \quad \text{(T-Proj)}$$

$$\frac{\{k_j:S_j \ ^{j \in 1..n}\} \text{ is a permutation of } \{l_i:T_i \ ^{i \in 1..n}\}}{\{k_j:S_j \ ^{j \in 1..n}\} <: \{l_i:T_i \ ^{i \in 1..n}\}} \quad \text{(S-RCDPERM)}$$

Example

$\{x{:}Nat, y{:}Nat\} <: \{x{:}Nat\}$ (S-RCDWIDTH)

$$\frac{0: Nat \qquad 0: Nat}{\vdash \{x=0, y=0\} : S=\{ x: Nat, y: Nat \}}\ (T\text{-}Tuple)$$

$$\frac{\vdash \{x=0, y=0\}: S \qquad S <: \{x : Nat\}}{}\ (T\text{-}SUB)$$

$$\frac{}{r : T' = \{x:\ T=Nat\}}\ (T\text{-}VAR)$$

$$\frac{}{r : \{x: Nat\} \vdash\ r : T' = \{\ldots\ x: T\ \ldots\}}\ (T\text{-}Proj)$$

$$\frac{r : \{x: Nat\} \vdash r.x : T}{}\ (T\text{-}ABS)$$

$$\frac{\vdash \lambda r : \{x: Nat\}.\ r.x : \{x: Nat\} \to T \qquad \vdash \{x=0, y=0\}: \{x : Nat\}}{}\ (T\text{-}APP)$$

$$\vdash (\lambda r : \{x: Nat\}.\ r.x)\ \{x=0, y=0\ \} : T$$

Properties of the type system(15.3)

- ~~Uniqueness of types~~
- Linear time type checking
- Type Safety
  - Well typed programs cannot go wrong
    - No undefined semantics
    - No runtime checks
  - If t is well typed then either t is a value or there exists an evaluation step $t \rightarrow t'$ [Progress]
  - If t is well typed and there exists an evaluation step $t \rightarrow t'$ then t' is also well typed [Preservation]

Upcasting (Information Hiding)

- Special case of ascription

$$\frac{\vdash t : S \qquad S <: T}{} \text{ (T-SUB)}$$

$$\frac{\Gamma \vdash t : T}{\Gamma \vdash t \text{ as } T : T} \text{ (T-ASCRIBE)}$$

Downcasting

- Generate runtime assertion
- But the typechecker can assume the right type

$$\frac{\Gamma \vdash t : S}{\Gamma \vdash t \text{ as } T : T} \quad \text{(T-DOWNCAST)}$$

~~$v \text{ as } T \rightarrow v$~~ ~~(E-ASCRIBE)~~

$$\frac{\vdash v : T}{v \text{ as } T \rightarrow v} \quad \text{(E-DOWNCAST)}$$

Progress is no longer guaranteed

Can replace downcasts by dynamic type check

$$\frac{\Gamma \vdash t_1 : S \qquad \Gamma, x: T \vdash t_2 : U \qquad \Gamma \vdash t_3 : U}{\Gamma \vdash \text{ if } t_1 \text{ in } T \text{ then } x \mapsto t_2 \text{ else } t_3 : U} \quad \text{(T-TYPETEST)}$$

$$\frac{\vdash v : T}{\text{if } v \text{ in } T \text{ then } x \mapsto t_2 \text{ else } t_3 \rightarrow [x \mapsto v]\, t_2} \quad \text{(E-TYPETESTT)}$$

$$\frac{\nvdash v : T}{\text{if } v \text{ in } T \text{ then } x \mapsto t_2 \text{ else } t_3 \rightarrow t_3} \quad \text{(E-TYPETESTF)}$$

Downcasting vs. Polymorphism

- Downcasting is useful for reflection
- Polymorphism saves the need for downcasts in most cases
  - reverse : $\forall$X: list X $\rightarrow$ list X
- Polymorphism leads to shorter and more secure code
- Polymorphism can have better performance
- Downcasting complicates the runtime system
- Polymorphism complicates language definition
- The interactions between polymorphism and subtyping complicates type checking/inference
  - Irrelevant for Java

Variants

Modified syntactic forms

t ::= …. Terms:

<l=t> ~~as T~~ tagging

case t of $<l_i = x_i> \Rightarrow t_i^{\ i \in 1..n}$ case

v ::= …. Values:

<l=v> ~~as T~~ tagged value

T ::= …. types:

$<l_i = T_i^{\ i \in 1..n}>$ type of variants

Evaluation Rules extends $\rightarrow$

$$\frac{t_i \rightarrow t'_i}{<l_i=t_i> \text{ ~~as T~~} \rightarrow <l_i=t'_i> \text{ ~~as T~~}} \quad \text{(E-VARIANT)}$$

$$\text{case } (<l_j = v> \text{ as T}) \text{ of } <l_i = x_i> \Rightarrow t_i^{\ i \in 1..n} \rightarrow [x_j \mapsto v]\, t_j \quad \text{(E-CaseVariant)}$$

$$\frac{t \rightarrow t'}{\text{case t of } <l_i = x_i> \Rightarrow t_i^{\ i \in 1..n} \rightarrow \text{case t' of } <l_i = x_i> \Rightarrow t_i^{\ i \in 1..n}} \quad \text{(E-CASE)}$$

Modified Type rules for Variants

Modified syntactic forms

t ::= ....
<l=t> ~~as T~~
case t of $<l_i = x_i> \Rightarrow t_i^{\ i \in 1..n}$

Terms:
tagging
case

v ::= ....
<l=v> ~~as T~~

Values:
tagged value

T ::= ....
$<l_i = T_i^{\ i \in 1..n}>$

types:
type of variants

modified typing rules

$$\frac{\Gamma \vdash t_j : T_j}{\Gamma \vdash <l_j = t_j> \text{ ~~as } <l_i = T_i^{\ i \in 1..n}>} \ : \ <l_j = T_j>$$ (T-VARIANT)

For each i $\Gamma, x_i : T_i \vdash t_i : T$
$\Gamma \vdash t : <l_i = T_i^{\ i \in 1..n}>$

case t of $<l_i = x_i> \Rightarrow t_i^{\ i \in 1..n} : T$ (T-CASE)

New subtyping rules

$<l_i:T_i^{\ i \in 1..n}> <: <l_i:T_i^{\ i \in 1..n+k}>$ (S-VARIANTWIDTH)

$$\frac{\text{For each i } S_i <: T_i}{<l_i:S_i^{\ i \in 1..n}> <: <l_i:T_i^{\ i \in 1..n}>}$$ (S-VARIANTDEPTH)

$$\frac{<k_j:S_j^{\ j \in 1..n}> \text{ is a permutation of } <l_i:T_i^{\ i \in 1..n}>}{<k_j:S_j^{\ j \in 1..n}> <: <l_i:T_i^{\ i \in 1..n}>}$$ (S-VARIANTPERM)

Lists

$$\frac{S <: T}{\text{List } S <: \text{List } T}\ \text{(S-List)}$$

References

$$\frac{S <: T \qquad T <: S}{\text{ref } S <: \text{ref } T} \text{ (S-Ref)}$$

Arrays

$$\frac{S <: T \qquad T <: S}{\text{Array } S <: \text{Array } T} \text{ (S-Array)}$$

$$\frac{S <: T}{\text{Array } S <: \text{Array } T} \text{ (S-ArrayJava)}$$

Base Types

Bool <: Nat (S-BoolNat)

Coercion Semantics for Subtyping (15.6)

- The compiler can perform conversions between types
  - Sometimes at compile time
- Subtyping is no longer transparent
- Improve performance

Summary

- Subtyping improves reuse
- Tricky semantics
- Complicates type checking/inference
- Well understood for many language features