中国科学技术大学
University of Science and Technology of China

Subtyping
(Dynamic Polymorphism)
《程序语言设计和程序分析》

张昱

yuzhang@ustc.edu.cn

中国科学技术大学

计算机科学与技术学院

References

- PFPL
  - Chapter 24 Structural Subtyping
  - Chapter 27 Inheritance
- TAPL (pdf)
  - Chapter 15 Subtyping
- [Concepts in PLs]

Subtyping and Inheritance

- **Interface**
  - The external view of an object
- **Subtyping**
  - Relation between interfaces
- **Implementation**
  - The internal representation of an object
- **Inheritance**
  - Relation between implementations

Various Object-Oriented Languages

- **Pure dynamically-typed OO languages**
  - Object implementation and run-time lookup
  - Class-based languages (Smalltalk)
  - Prototype-based languages (Self, JavaScript)
- **Statically-typed OO languages**
  - C++
    - using static typing to eliminate search
    - problems with C++ multiple inheritance
  - Java
    - using Interfaces to avoid multiple inheritance

Smalltalk: Subtyping

- If interface **A** contains all of interface **B**, then **A** objects can also be used **B** objects.

Point	ColorPoint
x:y:	x:y:
moveDx:Dy:	moveDx:Dy:
x	x
y	y
draw	color
	draw

ColorPoint interface contains Point
ColorPoint is a subtype of Point

Subtyping and Inheritance

Smalltalk/JavaScript subtyping is implicit

- Not a part of the programming language
- Important aspect of how systems are built

Inheritance is explicit

- Used to implement systems
- No forced relationship to subtyping

- **C++ is an object-oriented extension of C, Bell Labs**
- **Object-oriented features**
  - Classes
  - Objects, with dynamic lookup of virtual functions
  - Inheritance
    - Single and multiple inheritance
    - Public and private base classes
  - Subtyping
    - Tied to inheritance mechanism
  - Encapsulation
    - Public, private, protected visibility

C++: Virtual functions

- **Member functions are either**
  - Virtual, if explicitly declared or inherited as virtual
  - Non-virtual otherwise
- **Virtual functions**
  - Accessed by indirection through ptr in object
  - May be redefined in derived (sub) classes
- **Non-virtual functions**
  - Are called in the usual way. *Just ordinary functions*.
  - Cannot redefine in derived classes (except overloading)
- **Pay overhead only if you use virtual functions**

C++ Subtyping

- **Subtyping in principle**
  - A <: B if every A object can be used without type error whenever a B object is required
- **C++: A <: B if class A has public base class B**
  - Independent classes not subtypes

Java

- **1990-95 James Gosling and others at Sun**
- **Syntax similar to C++**
- **Object**
  - has fields and methods
  - is allocated on heap, not run-time stack
  - accessible through reference (only ptr assignment)
  - garbage collected
- **Dynamic lookup**
  - Similar in behavior to other languages
  - Static typing => more efficient than Smalltalk
  - Dynamic linking, interfaces => slower than C++

Inheritance

- **Similar to Smalltalk, C++**
- **Subclass inherits from superclass**
  - Single inheritance only (but Java has interfaces)
- **Some additional features**
  - Conventions regarding *super* in constructor and *finalize* methods
  - Final classes and methods cannot be redefined

Interfaces vs Multiple Inheritance

C++ multiple inheritance

- A single class may inherit from two base classes
- Constraints of C++ require derived class representation to resemble *all* base classes

Java interfaces

- A single class may implement two interfaces
- No inheritance (of implementation) involved
- Java implementation (discussed later) does not require similarity between class representations

Subtyping Principles

- **Subtyping judgement** $\tau' <: \tau$
  - $\dfrac{}{\tau <: \tau}$ $\qquad$ $\dfrac{\tau'' <: \tau' \quad \tau' <: \tau}{\tau'' <: \tau}$
- **Subsumption rule**

$$\frac{\Gamma \vdash e : \tau' \quad \tau' <: \tau}{\Gamma \vdash e : \tau}$$

- **Numeric types**
  - $\text{int} <: \text{rat} <: \text{real}$
- **Product types，Sum types**

$$\frac{J \subseteq I}{\langle \tau_i \rangle_{i \in I} <: \langle \tau_j \rangle_{j \in J}} \qquad \frac{J \subseteq I}{[\tau_i]_{i \in I} <: [\tau_j]_{j \in J}}$$

Width subtyping

(较宽积类型是较窄积类型的子类型)

Subtyping Principles

Variance:

- Product and sum types: Depth subtyping (Covariance)

$$\frac{\tau_i' <: \tau_i (\forall i \in I)}{\langle \tau_i' \rangle_{i \in I} <: \langle \tau_i \rangle_{i \in I}} \qquad \frac{\tau_i' <: \tau_i (\forall i \in I)}{[\tau_i']_{i \in I} <: [\tau_i]_{i \in I}}$$

- Partial function types
  - covariant in its range.

$$\frac{\tau_2' <: \tau_2}{\tau_1 \rightharpoonup \tau_2' <: \tau_1 \rightharpoonup \tau_2}$$

  - contravariant in its domain position

$$\frac{\tau_1' <: \tau_1}{\tau_1 \rightharpoonup \tau_2 <: \tau_1' \rightharpoonup \tau_2}$$

Subtyping Principles

Quantified Types

$$\frac{\Delta, t \text{ type} \vdash \tau' <: \tau}{\Delta \vdash \forall(t.\tau') <: \forall(t.\tau)} \qquad \frac{\Delta, t \text{ type} \vdash \tau' <: \tau}{\Delta \vdash \exists(t.\tau') <: \exists(t.\tau)}$$

- Substitution： If $\Delta, t \text{ type} \vdash \tau_1 <: \tau_2$, and $\Delta \vdash \tau \text{ type}$, then $\Delta \vdash [\tau/t]\tau_1 <: [\tau/t]\tau_2$

THANKS