Typy generyczne wyższych rzędów

Zagadnienia Programowania Obiektowego

Piotr Wojnarowski

Wydział Matematyki, Informatyki i Mechaniki
Uniwersytet Warszawski

26.04.2010

Spis treści

Generics of a Higher Kind, OOPSLA 2008

- Adriaan Moors, Frank Piessens- Katolicki Uniwersytet w Leuven
- Martin Oderksy- Ecole Politechnic w Lozannie

Typy generyczne pierwszego rzędu

Czym są typy generyczne (uogólnione)?

- Klasyczny przykład: List<T>
- Dostępne nie od dziś w wielu językach programowania: języki funkcyjne, Java, C#, C++
- Uogólnienie kodu, niezależnie od klas, redukcja duplikacji kodu

Kilka definicji

- Typ konkretny, np: List[Int]
- Konstruktor typu uogólnionego, np: List
- Deklaracja typu uogólnionego, np: List[T]

A gdyby można było uogólnić bardziej?

Przyład

```
trait Iterable[T] {
  def filter(p: T => Boolean): Iterable[T]
  def remove(p: T => Boolean): Iterable[T] =
filter (x => !p(x))
}
trait List[T] extends Iterable[T] {
  def filter(p: T => Boolean): List[T]
  override def remove(p: T => Boolean): List[T] =
filter (x => !p(x))
}
```

Można! (W Scali)

Przyład

```
trait Iterable[T, Container[X]] {
def filter(p: T => Boolean): Container[T]
def remove(p: T => Boolean): Container[T] =
filter (x => !p(x))
}
trait List[T] extends Iterable[T, List]
```

Typy wyższych rzędów w skrócie

- Możemy mieć uogólnione konstruktory typów
- Jak widać, pozwala to na dalszą redukcję kodu
- Dostępne w językach takich jak Haskell czy Scala
- Praktyczne zastosowania: list comprehensions (budowanie list na podstawie innych list, operator yield), kombinatory parserów, języki dziedzinowe (DSL)

Krótkie przypomnienie składni

- trait- interface + mixin
- object- singleton, jak klasa
- pola typów (type members):

Type member

```
trait List{
type T
}
List{type T=Int}
```

Dalsze możliwości

Ograniczanie parametrów:

Iterowalna kolekcja

```
trait Iterable[T] {
type Container[X] <: Iterator[T]
}
```

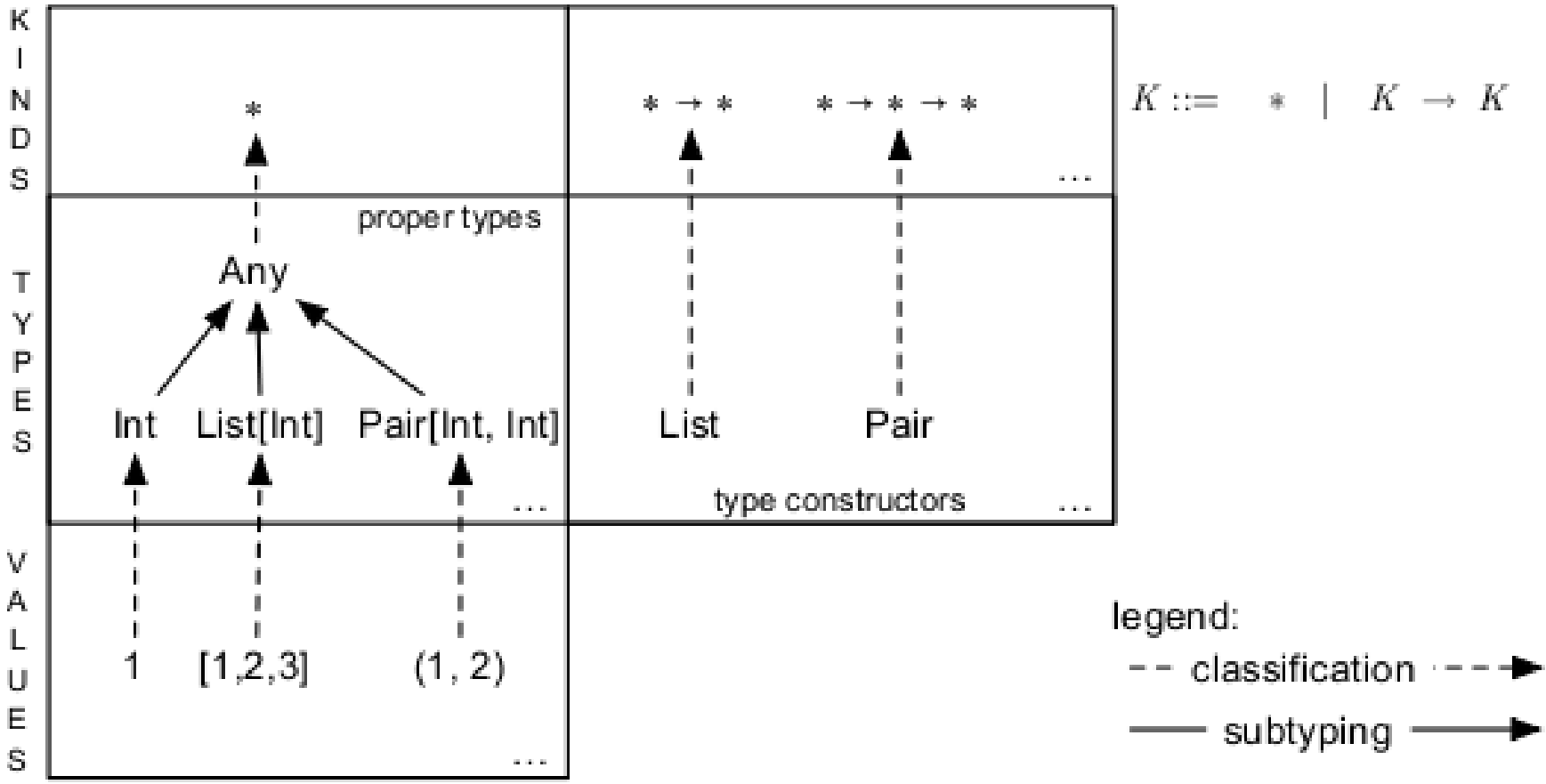
Analogicznie >:

In-, ko- i kontrawariancja

Typy uogólnione nie muszą „dziedziczyć” w zwykłej kolejności

- Stack[T] jest podtypem Stack[S] wtw S == T
- Stack[+T] jest podtypem Stack[S] wtw S >: T
- Stack[-T] jest podtypem Stack[S] wtw S <: T

Rodzaje- schemat



Po co rodzaje?

- Rodzaje- konstrukcja niedostępna dla programisty, użyta na poziomie kompilatora
- *- wartości
- -> - konstruktor typów, klasyfikuje konstruktory typów
- * (T, U) - typ o dolnym ograniczeniu T i górnym U
- Nothing, Any- domyślne wartości dla T i U, * (Nothing, Any) == *, *(Nothing, U) == * (U)
- Container[X] <: Iterable[X] == X @ * -> * (Iterable[X])
- Pod- rodzaje: * (T, U) <: * (T', U') wtw gdy T' <: T i U <: U'

Po co rodzaje, przykład

Iterowalna lista numeryczna

```
class Iterable[Container[X], T]
trait NumericList[T <: Number] extends
    Iterable[NumericList, T]
```

- Błąd podczas kompilacji! Dlaczego?
- NumericList: * (Number) -> *
- Container: * -> *

Po co rodzaje, przykład

Iterowalna lista numeryczna

```
class Iterable[Container[X], T]
trait NumericList[T <: Number] extends
    Iterable[NumericList, T]
```

- Błąd podczas kompilacji! Dlaczego?
- NumericList: * (Number) -> *
- Container: * -> *

Po co rodzaje, poprawnie

Iterowalna, ograniczona lista

```
class Iterable[Container[X <: Bound],
T <: Bound, Bound]
trait NumericList[T <: Number] extends
    Iterable[NumericList, T, Number]
```

Parametry domniemane

implicits

```
trait Ord[T] {
    def <= (other: T): Boolean
}
import java.util.Date
implicit def dateAsOrd(self: Date)
  = new Ord[Date] {
    def <= (other: Date) = self.equals(other)
|| self.before(other)
}
def max[T <% Ord[T]](x: T, y: T): T
= if(x <= y) y else x
```

Ograniczenia

Czego nie da się zrobić w Scali?

- Częściowa aplikacja parametrów typów
- Funkcje anonimowe z typami uogólnionymi
- Currying
- Brak interfejsu typów dla typów uogólnionych wyższych rzędów

Przynajmniej na razie

Podsumowanie

- Redukcja duplikacji kodu w bibliotekach wykorzystujących typy uogólnione
- Łatwiejsze używanie takich bibliotek, np. nie trzeba rzutować kolekcji otrzymanej z filter() na żądany typ
- Sprzyjają pisaniu bilioteki z ich użyciem tak, żeby użytkownik nawet nie wiedział o ich zastosowaniu, ale zauważył łatwiejszą obsługę

Koniec

Dziękuję