Pizza into Java: Translating theory into practice

Martin Odersky

Philip Wadler

Outline of the talk

- Introduction
- Three features of Pizza and their implementations
  - . Parametric polymorphism
  - . Higher-order functions
  - . Algebraic data types
- Discussion
  - . Typing issues
  - . Rough edges
- Conclusion

Introduction

- Pizza and Java

  --- Pizza is a strict superset of Java

  . Why a strict superset?

  --- Three academic ideas:

  . Parametric polymorphism

  . Higher-order functions

  . Algebraic data types

Introduction

- Translating Theory into practice

  --- Heterogeneous Translation

  A specialized copy of code for each type it is used at.

  --- Homogenous Translation

  A single copy of code with a universal representation.

- Status

  --- A preliminary design, including typing rules

  --- A pizza compiler in Pizza is free available on the web.

  --- GJ, a more advanced PL to add generic functions to Java.

Parametric polymorphism

- Parametric polymorphism

  --- [Strachey 67]Parametric polymorphism is obtained when a function works on a range of types, these types normally exhibit some common structure.

  --- The uniformity of type structure is achieved by type parameters, implicit or explicit.

  --- In addition to functions, classes and interfaces can also be parameterized.

Parametric Polymorphism

- Why parametric polymorphism?

  --- Reuse

  --- Compactness

  --- Safety

  --- Expressiveness

  e.g. Interface specification.

- Issues of parametric polymorphism

  --- Design problems

  . Type checking

  . Binding time

  --- Legacy problems

Parametric Polymorphism

- Forms of Polymorphism

-- Parametric polymorphism: A type variable X is bound to any type T.

-- Bounded polymorphism: A type variable X is bound to any subtype of a particular (parameterized) type P and X does NOT occur free in P.

-- F- Bounded polymorphism: A type variable X is bound to any subtype of a particular parameterized type P, and X occurs free in P, i.e. P=F(X) where F is a type functional.

Parametric Polymorphism

- Forms of Polymorphism

-- In F-Bounded polymorphism, X is often bound to a recursive type.

-- The reason why we need F-bounded polymorphism in addition to bounded polymorphism is that bounded polymorphism is not so flexible under the subtyping rules if P is a recursive type, and thus we need a more general bound for X. P=F(X) is the choice, and we can regard it as an interface specification.

(This Slide is added after the talk)

Parametric Polymorphism

- Java programmers' emulation

  1) Cut and paste codes,then specialization

  2) Eliminate type parameters, bind type variables to Object, cast types when needed, also add bridges when needed

Parametric Polymorphism

- Pizza's implementation
  Parametric polymorphism
  (Example 2.1)

  --- Heterogenous translation
  (Example 2.2)
  --- Homogenous translation
  (Example 2.3)

Parametric Polymorphism

- Pizza’s implementation

Bounded Parametric polymorphism

(Example 2.4)

--- Heterogenous translation

(Example 2.5)

--- Homogenous translation

(Example 2.6)

Parametric Polymorphism

- Arrays

  --- An array could be regarded as a parameterised class.

  --- Features of Java's array

  . Polymorphism is achieved through subtyping, i.e. A is a subtype of B => A[] is a subtype of B[]. (Unsafe, and run-time checks are required.)

  e.g. Animal[] x; Lamb[] lamb_flock = new Lamb[100]; x = lamb_flock; x[i] = new Wolf;

Parametric Polymorphism

- Arrays (Continued)

--- Features of Pizza's array

. Polymorphism is achieved through instantiation instead of subtyping. E.g. to match String[] to elem[]. (elem is a class variable),

. However, in order to be a super set of Java, the subtyping relation between arrays in Java is retained.

. Polymorphic array creation is probibited.

Parametric Polymorphism

- Pizza's implementation

  Parametrised arrays

  (Example 2.7)

  --- Homogenous translation

  (Example 2.8 and 2.9)

Higher-order functions

- First-class functions

  --- Call by name (Algol 60)

  .So-called "Algol Wall" : program structure <> object strcuture

  --- Lambda expressions (Lisp)

  . Classical Lisp design violates "orthogonality"

  . Use "function" and "quote" to pass/return functions

Higher-order functions

- First-class functions(Continued)

--- Closure (Scheme)

. A closure C = a “Lambda” expression L + an environment E.

. A closure is a time-capsule,which is different from dynamic binding.

. Dynamic binding(scoping) does not need closures.

Higher-order functions

- Classes VS closures

  Both break the Algol 60 wall. Which is better?

  --- Higher-order functions can be implemented as objects.

  -- Classes naturally lead to modularity.

  --- But sometimes closures are more convenient.

Higher-order Functions

- Pizza's first-class functions
  --- Syntax

  func type: (t1,…,tn) -> t0
  [Why not use to(t1,…,tn) as func type?]

  func instance:
  fun (t1 x1,…,tn xn)-> t0 s

Higher-order Functions

- Pizza's first-class functions

  --- Semantics

  - Three sorts of variables

    ---formal parameters P

    ---free variables F

    ---instance variables I

  - Conceptually,

  --- P (pass by value)

  --- I (pass by reference)

  --- F (pass by reference, and may pass by value after analysis)

Higher-order Functions

- Pizza's first-class functions

  --- Semantics

  (Example 3.1)

  . Heterogenous Translation

  (Example 3.2)

  . Homogenous Translation

Algebraic types

- Algebraic types

  --- Constructors

  . A constructor produces a member of the type, we can also think that a constructor produces a type, the whole type being defined is a union of the constructed types.

Algebraic Types

- Algebraic types

--- Pattern match

. A pattern p is either a variable v,or a constant k, or a constructor pattern, of the form (c p1 … pr) where c is a constructor of arity r, and p1,…,pr are patterns.

.What is a match? In other words, what is a pattern-match lambda abstraction?

Algebraic Types

- Pizza's algebraic types
  (Example 4.1)
  . Heterogenous Translation
  (Example 4.2)

- A integrated example of polymorphism, higher-order functions and algebraic types. (Example 4.3)

Discussion

- Typing issues

  --- Integrate subtyping and parametric polymorphism

  . Subsumption , Complete and matching

  . Covariance and Contra-variance

Discussion

- Typing issues

--- Integrating dynamic typing

. Dynamic typing: new types can be defined/created at run-time. (Related concepts: dynamically typed and dynamic type.).

. Type checking existential types.

Discussion

- Rough edges

  --- Casting

  --- Visibility

  --- Dynamic loading

  --- Interfaces for built-in classes

  --- Tail calls

  --- Arrays

Conclusion

A natural translation technique with only few rough edges has been exploited by Pizza, a language with a type--driven design, to translate three well-known theoretic features in functional programming system to Java, a strict subset of Pizza. However, the practical effect is still to wait and see.