Generic Programming in F#

Datatype generic programming for .Net

Ernesto Rodriguez
Utrecht University
e.rodriguez@students.uu.nl

Wouter Swierstra
Utrecht University
W.S.Swierstra@uu.nl

Abstract

The introduction of Datatype Generic programming (DGP) *revolutionized* functional programming by allowing numerous algorithms to be defined by induction over the structure of types while still providing type safety. Due to the advanced type system requirements for DGP, only a handful of functional languages can define generic functions making it inaccessible to most programmers. Ordinary languages provide reflection and duck typing as a mechanism to specify generic algorithms. These mechanisms are usually error prone and verbose. By combining ideas from DGP and implementing them through reflection, a type-safe interface to DGP has been built for the F# language. These generic algorithms can be accessed by any language running in the .Net platform.

Categories and Subject Descriptors CR-number [*subcategory*]: third-level

General Terms term1, term2

Keywords keyword1, keyword2

1. Introduction

2. The F# Language

The F# programming language is a functional language of the ML family. It focuses on being a productive language by levering on functional programming and at the same time easy to adopt by programmers of other .Net hosted languages. As a result, the lenguage has a much simpler type system than Haskell or Scala. Most of the development effort in the language has focused on features to work with data (like type-providers) and to be compatible with the .Net type system. Unlike Scala, F# performs no type errasure when compiled to the .Net platform.

There are several mechanism to define new types in F#: classes, records and algebraic data types. Classes correspond to the traditional object oriented paradigm and are allowed to inherit fields and functions from another type as long as the type is not sealed (which is a .Net attribute for types). Records and algebraic datatypes correspond to the functional approach of defining types. Records and ADTs are always sealed and can be pattern matched. All types in F# can define member functions (methods) and can implement any number of interfaces. Types can also have generic type arguments but they are required to be of kind $*$ (star).

3. The .Net platform

The .Net platform is a common runtime environment to allow the execution of a family of languages. It implements a very rich type system which includes support for generics. Many type operations that happen in F# (such as sub-typeing) are handled by the .Net platform. The sub-typeing relation will be denoted by $\tau_a :> \tau_b$ which means τ_a is a sub-type of τ_b and consequently a value of type τ_a can be automatically converted to a value of type τ_b.

Like most object oriented langagues, .Net sub-typeing mechanism that allows types to be automatically converted to types which are higher in the class hierarchy. A well known restriction of this mechanism is that sub-typeing rules cannot automatically be applied to generic type arguments. In other words $\tau_a :> \tau_b \not\Rightarrow T < \tau_a > :> T < \tau_b >$.

The F# language does not support generics of higher kind. This means that generic types in F# cannot be applied to other types. This feature is used by other DGP [**? ? ? ?**] to enforce type safety and allow the compiler to select

A. Appendix Title

This is the text of the appendix, if you need one.

Acknowledgments

Acknowledgments, if needed.

References

[1] P. Q. Smith, and X. Y. Jones. ...reference text...

CONF 'yy, Month d–d, 20yy, City, ST, Country.

http://dx.doi.org/10.1145/nnnnnnn.nnnnnnn