Dependent Types in Scala

Yao Li
@lastland0

Static type systems are the world’s most successful application of formal methods. Types are simple enough to make sense to programmers; they are tractable enough to be machine-checked on every compilation; they carry no run-time overhead; and they pluck a harvest of low-hanging fruit.

–-- Brent A. Yorgey, et al. Giving Haskell a Promotion.

Types Are Specifications

```
sealed abstract class List[+A] {

 def apply(n: Int): A

}
```

Types Are Specifications

- null vs. None
  - Tony Hoare’s “billion-dollar mistake”
  - Option types enforce programmers to check the “nullness” of a value in their implementation

Types Are Simple

```
scala> val l = List(1, 2, 3)
l: List[Int] = List(1, 2, 3)

scala> l(3)
java.lang.IndexOutOfBoundsException: 3
  at scala.collection.LinearSeqOptimized.apply(LinearSeqOptimized.scala:63)
  at scala.collection.LinearSeqOptimized.apply$(LinearSeqOptimized.scala:61)
  at scala.collection.immutable.List.apply(List.scala:86)
  ... 30 elided
```

Types Are Simple

- They can not specify a red-black tree.
- Invariants of a red-black tree:
  - Each node is either red or black.
  - The root is black.
  - All leaves are black.
  - If a node is red, then both its children are black.
  - Every path from a given node to any of its descendant leaves contains the same number of black nodes.

Dependent Types

```
Fixpoint rep (A: Type) (n: nat) (a: A):
    Vector A n:=
    match n with
    | O => VNil A
    | S n’ => VCons A n’ a (rep A n’ a)
    end.
```

Roadmap

- In this talk, we will implement the following in Scala:
    - a vector whose length information is in its type,
    - a rep function which takes a number, and returns a Vector of exactly that length,
    - an app function that takes two vectors, and returns a list whose length is the sum of their lengths,
    - and an indexing method with compile-time bounds checking.

Demo

There's more!

Dependent Types

- Theorem prover.
  - Propositions as types!
- Formal verification.
  - For further reading: *Verified Functional Algorithms,* by Andrew Appel.
- Certified softwares.
  - CompCert, VST, CertiKOS, FSCQ, Kami, etc.

Further Reading

- Full dependent type languages:
  - Gallina (Coq)
    - Software Foundations, by Benjamin C. Pierce
  - Idris
    - Type-Driven Development with Idris, by Edwin Brady
  - Agda
  - …

Further Reading

- Dependent Types in Haskell
  - Dependently Typed Programming with Singletons, by Richard Eisenberg and Stephanie Weirich
  - Depending on Types, by Stephanie Weirich (https://www.youtube.com/watch?v=n-b1PYbRUOY)
  - The Influence of Dependent Types, by Stephanie Weirich (https://www.youtube.com/watch?v=GgD0KUxMaQs)

References

- All the codes I have shown are written by myself, but I would not know how to write them without the help of following materials:
  - Dave Gurnell's *The Type Astronaut's Guide to Shapeless.*
  - The source code of shapeless library: https://github.com/milessabin/shapeless/
  - Miles Sabin's demo at StrangeLoop 2013: https://github.com/milessabin/strangeloop-2013
  - Miles Sabin's dependently typed red-black tree: https://github.com/milessabin/tls-philly-rbtree-2016

Q&A

Thanks!

Source code at: https://github.com/lastland/DTScala