Meeting 06 - Inductive Data Types

Bor-Yuh Evan Chang

Thursday, September 12, 2024

Meeting 06 - Inductive Data Types

What questions does your neighbor have?

In-Class Slides

In-Class Jupyter

Book Chapter

Announcements

- Lab 1 due this Friday 9/13 6pm
  - Use GitHub and VS Code: Submit `Lab1.scala`. Just read Jupyter notebook or use it for scratch work.
  - No autograder on Gradescope. Submit via GitHub.
  - Take advantage of lab section Friday to finish!

Today

- Inductive Data Types
- Triage Your Questions
  - Lab 1?
- Revisit and Go Deeper On:
  - Recursion (Meeting 05)
  - Data Types (Meeting 04), if time permits
  - Binding and Scope (Meeting 03), if time permits

Questions?

- Review:
  - What's *tail recursion*? How does it relate loops? (Krin)

- Defining an "inner function" (divia)
- Functional vs Imperative Prog (Ojas)
  - Immutable
  - Mutation

(1 + 2) → 3

((x: Int) ⇒ x+1) (3) → 3+1 → 4

Questions?

MyList

A list is an *inductive data type*:

```
defined object MyList
```

Recursion on Lists

```
defined function length
```

Without Pattern Matching

```
defined function length
```

Could `l.tail` fail?

Append

```
defined function append
```

Now, we see why `append` (and `:::` in the Scala library) has to be linear time.

Buggy Append

What does buggyAppend do?

```
defined function buggyAppend
```

Reverse

```
defined function reverse
```

Exercise: Tail-Recursive Reverse

```
defined function reverse
```

Trees

```
:
Compilation Failed
```

Persistent Data Structures

```
m: Map[Int, List[String]] = Map(
  2 -> List("two", "dos", "\u4e8c"),
  10 -> List("ten", "diez", "\u5341")
)
newm: Map[Int, List[String]] = Map(
  2 -> List("two", "dos", "\u4e8c"),
  10 -> List("ten", "diez", "\u5341"),
  14 -> List("fourteen", "catorce", "\u5341\u56db")
)
```

An Object Language

Since we want to study small sub-languages common to programming languages in general, we don't care much about concrete syntax.

Let's consider JavaScripty to be the subset of JavaScript we want to study at the moment, for example, number literals and `+`:

Our goal is to implement semantics following JavaScript (for this small JavaScripty).

Parsing and Abstract Syntax

The process of converting a program in concrete syntax (i.e., as a string) to a program in abstract syntax (i.e., as a tree) is called *parsing*.

Abstract Syntax Trees (ASTs)

```
:
Compilation Failed
```

Eval

```
defined function eval
```