Lecture 5: Haskell

CS 181
Spring 2016
Kim Bruce

Some slide content taken from Unger and Michaelis

Types in Linguistics

- Categorial Grammars introduced by Montague.
- Def: CAT, the set of categories is the smallest set such that:
  1. S, CN, and IV are in CAT
  2. If A and B are in CAT, so is A/B
- Intuition for type A/B is a term that, if provided with one of type B, would result in a term of type A.

Categorial Definitions:

Definition	*Description*	*Expressions*
S	Sentences	
CN	Common nouns	man, woman, ball, ...
IV	Intransitive verb phrases	walk, talk, sleep, ...
T = S/IV	Terms (noun phrases)	John, Mary, he, she, ...
TV = IV/T	Transitive verb phrases	love, take, throw, ...
IV/S	Sentential complement verbs	believe that, know that
IV/IV	Infinital complement verbs	try to, wish to,
CN/CN	Prenominal adjectives	red, small, fat, ...
S/S	Sentence-modifying adverbs	necessarily, ...
T/CN	Determiners	every, some, the, one, ...

Rephrase as Function Types

- A/B = B → A
- For example:
  - T = IV → S
  - TV = T → IV = (IV → S) → IV
  - Adj = CN → CN
  - Det = CN → T
  - ...
- Use to classify phrases

Why Types?

- Types help us to interpret linguistic phrases
  - as well as to rule some out as ill-formed
- If types make sense, then can provide tremendous help in specifying semantics and figuring out meaning.
- To help, want to write programs to compute meanings of phrases.
- Best if language reflects formal model

Haskell for Semantics

- Purely functional (& lazy) language created about 1990 to further research in functional programming (as well as writing applications)
- Built on ideas of Miranda© and ML.
- Statically and strongly typed.
  - Same type inference as ML.

Functional Thinking

- Identifiers refer to immutable values.
  - No variables or assignments.
- Obtain results by pushing values through pipeline of transformations.
- Main program is function with program's input as argument.
- The main function is defined as a composition of helper functions which are themselves defined from other functions.

Why Haskell

- Haskell is a good choice for this course because we will be defining the semantics of complex language expressions as higher-order functions.
- Haskell is also based on lambda calculus. Therefore the shift from formal semantics to implementation is very small.

Getting Started

- You can get started by downloading the Haskell platform from https://www.haskell.org/platform/. We will be using Haskell 7.10.3 in this class.
- Installing this package will provide you with the Glascow Haskell compiler and its associated libraries.

Starting Haskell

- The ghci command can be used to run a Haskell interpreter.

```
Prelude> 2 + 3
5
Prelude> True && False
False
Prelude> (3+7) * 5
50
```

- Functions are best written in a separate file.

Writing Programs

- Write the following code to a text file and save it as first.hs:

```
double :: Int -> Int
double n = 2 * n
```

- Inside GHCi, you can load the program with
  :load first.hs

Using GHC

- to enter interactive mode type: ghci
  - :load myfile.hs -- :l also works
  - after changes type :reload myfile.hs
  - Type :q or Control-d to exit
  - :set +t -- prints more type info when interactive
  - "it" is result of expression

May need to add /Library/Haskell/bin to Path

Using a program

- Once loaded you can use a program as you like
  - double 17
  - double (5×3)
  - double (double 17)
- You can use :t to inquire as to an expression's type:
  - :t double
    double :: Num a => a -> a
  - :q exits from the interactive environment.

Learning Haskell

- Recommend the online text Learn you a Haskell for greater good.
  - The title is stupid but the text is actually quite good.
- I also recommend "10 things you should know about Haskell syntax"

Built-in data types

- Unit has only ()
- Bool: True, False with not, &&, ||
- Int: 5, -5, with +, -, *, ^, =, /=, <, >, >=, ...
  - div, mod defined as prefix operators (`div` *infix*)
  - Int fixed size (usually 64 bits)
  - Integer gives unbounded size
- Float, Double: 3.17, 2.4e17 w/ +, -, *, /, =, <, >, >=, <=, sin, cos, log, exp, sqrt, sin, atan.

More Basic Types

list of Char

- Char: 'n'
- String = [Char], not really primitive
  - "hello"++" there", length
  - No substring, but `isInfixOf` for all lists
  - Also 'isPrefixOf`, `isSuffixOf'

Prefix op w/out `!

import Data.List

Interactive Programming with ghci

- Type expressions and run-time will evaluate
- Define abbreviations with "let"
  - let double n = n + n
  - let seven = 7
- "let" not necessary at top level in programs loaded from files
- Comments start w/ -- and go to end of line

Lists

- Lists
  - [2,3,4,9,12]: [Integer]
  - [] -- empty list
  - Must be homogenous
  - Functions: length, ++, :, map, rev
    - also head, tail, *but normally don't use!*

Polymorphic Types

- [1,2,3]:: [Integer]
- ["abc", "def"]:: [[Char]], ...
- []:: [a]
- map:: (a → b) → ([a] → [b])
- *Use* :t exp *to get type of* exp

Pattern Matching

- Decompose lists:
  - [1,2,3] = 1:(2:(3:[]))
- Define functions by cases using pattern matching:

```
prod [] = 1
prod (fst:rest) = fst * (prod rest)
```

Pattern Matching

- Desugared through case expressions:
  - ```
    head' :: [a] -> a
    head' [] = error "No head for empty lists!"
    head' (x:_) = x
    ```
- equivalent to
  - ```
    head' xs = case xs of
              [] -> error "No head for empty lists!"
              (x:_) -> x
    ```

Type constructors

- Tuples
  - (17,"abc", True) : (Integer , [Char] , Bool)
  - fst, snd defined only on pairs
- Records exist as well
  - read about on your own

More Pattern Matching

- (x,y) = (5 \`div\` 2, 5 \`mod\` 2)
- hd:tl = [1,2,3]
- hd:_ = [4,5,6]
  - "_" is wildcard.

Questions?