Session 2: Types

COMP2221: Functional programming

Laura Morgenstern*

*laura.morgenstern@durham.ac.uk

Recap

- What are some differences between functional and imperative programming?
- Which programming model more closely mirrors the way computers execute?
- What are interpreters and compilers? (in *very* broad terms)
- What are side effects?
- Why can side effects easily introduce bugs?
- What is Haskell's syntax for identifier definition, function application and composition?
- Which list operations does the Prelude provide?

Definition: What are types?

Definition (Type)

A type is a *collection* of related values.

A type can be described by specifying

- the set of data elements it covers and
- the operations it supports.

Example

- `Bool` the two logical values `True` and `False`.
- `Integer -> Integer` the set of all functions that take an `Integer` as input and produce an `Integer` as output.

Motivation: Why do we need types?

Example in C/Java

```
int a = 4;
int b = 3;

double c = a/b;
```

```
double a = 4;
double b = 3;

double c = a/b;
```

Motivation: Why do we need types?

Example in C/Java

```
int a = 4;
int b = 3;

double c = a/b;
```

```
double a = 4;
double b = 3;

double c = a/b;
```

Result depends on input types.

Since computers represent *everything* as sequences of bits, types are also required to interpret these bit patterns.

Motivation: Why do we need types?

- Mathematics and programming rely on the notion of *types*
- Tell us *how* to interpret a variable
- Provide restrictions on valid *operations*
- Are required to know what a bit pattern means

Implementation

Find the correct implementation of an operator.

Correctness

Check whether an operation on some data is valid and/or well-defined.

Check whether a code fragment is correct (type safety).

Documentation

Document the code's semantics for the reader.

Type System Classification

- Strongly vs. weakly typed languages
- Statically vs. dynamically typed languages

Static type checking

Translators must check for *type correctness*

Definition (Statically typed language)

Type safety is checked at translation time.

⇒ invalid types result in translation error

```
-- Invalid
foo :: a -> Int
foo f = 1 + f
```

Dynamic type checking

Definition (Dynamically typed language)

Type safety is checked at run time.

⇒ invalid types only detected as soon as used

```python
# Fine as long as f supports addition with a number
def foo(f):
    return 1 + f
```

Type checking

- How does the translator determine the type of an expression?

Explicit annotation

Programmer annotates all variables with type information (e.g. C/Java)

Type inference

Translator *infers* the types of variables based on the operations used (e.g. Haskell)

Duck typing

Translator/runtime just tries the operation, if it succeeds, that was a valid type! (Python)

Types in Haskell

Haskell is

Strongly, statically typed.

$\Rightarrow$ every well-formed expression has exactly one type, these types are known at *compile time*

Notation and inspection

Attaching types

Haskell's notation for "e is of type T" is spelt

```haskell
e :: T
-- False is of type Bool
False :: Bool
-- not is of type Bool -> Bool
not :: Bool -> Bool
```

What type does X have?

Every valid expression in Haskell must have a valid type.

You can ask GHCi what the type of an expression is with the command `:type expr`

```
Prelude> :type sum
sum :: Num a => [a] -> a
```

Demo time

Let's look at some types

Functions have types

- Functions have types in all programming languages, Haskell makes this particularly explicit

Functions of one argument "unary"

Map from one type to another

```
not :: Bool -> Bool
and :: [Bool] -> Bool
```

Functions of two arguments "binary"

Map from two types to another

```
add :: (Int, Int) -> Int
```

"add eats two Ints and returns an Int"

Summary

- Content
  - Defined and motivated types
  - Different concepts of typing (dynamic/static)
  - Considered basic Haskell types
  - Looked at list and tuple types
  - Determined type of expressions with GHCi
  - Considered types of functions
- Self-study
  - Tackle exercise “Types and Lists”