Chapter 15

Functional Programming

Topics

- Values
- Functions
- Extensionality
- Currying
- Definitions

Values

- An expression is used to describe (or denote) a value.
  - Among the kinds of value are: numbers of various kinds, truth values, characters, tuples, functions, and lists.
  - New kinds of value can be introduced.
- The evaluator prints a value by printing its canonical representation.
  - Some values have no canonical representation (i.e. function values).
  - Other values are not finite (i.e. Π)

Values

- For some expressions the process of reduction never stops and never produces any result (i.e. the expression `infinity`).
- Some expressions do not denote well-defined values in the normal mathematical sense (i.e. the expression `1/0`).

- Every syntactically well-formed expression denotes a value.
  - A special symbol $\perp$ (bottom) stands for the undefined value of a particular type

Values

- The value of infinity is the undefined value $\perp$ or type `Integer`.
- `1/0` is the undefined value $\perp$ or type `Float`
  - `1/0 =` $\perp$
- The computer is not able to produce the value $\perp$.
  - It generates an error message or it remains perpetually silent.
- $\perp$ is a special value that can be added to the universe of values only if its properties and its relationship with other values are precisely stated.

Values

- If `f` $\perp$ `=` $\perp$, then `f` is *strict*, otherwise it is *nonstrict*.
- `square` is a strict function because the evaluation of the undefined value goes into an infinite reduction (i.e. `? square infinity`)
- `three` is nonstrict because the evaluation of the undefined value is `3` (i.e. `? three infinity`)

Functions

- A function `f` is a rule of correspondence that associates each element of given type `A` (domain) with a unique element of a second type `B` (range).
  - The result of applying function `f` to an element `x` of the domain is written as `f(x)` or `f x` (when the parentheses are not necessary).
    - Parentheses are necessary when the argument is not a simple constant or variable.

Functions

- Examples
  - `square(3+4)` vs. `square3+4`
    - `square3+4` means `(square3)+4`
  - `square(square3)` vs. `square square 3`
    - square square 3 means (square square) 3

Extensionality

- Two functions are equal if they give equal results for equal arguments.
  - `f = g` if an only if `f x = g x` for all `x`
  - This is called the principle of *extensionality*.
  - Example:

```
double, double' :: Integer → Integer
double x = x + x
double' x = 2 * x
```

  - `double` and `double'` defines the same functional value, `double = double'`

Currying

- Replacing a structure argument by a sequence of simpler ones is a way to reduce the number of parentheses in an expression.

```
smaller :: (Integer,Integer) → Integer
smaller (x,y) = if x ≤ y then x else y

smallerc :: Integer → (Integer → Integer)
smallerc x y = if x ≤ y then x else y
```


Currying: advantages

1. Currying can help to reduce the number of parentheses that have to be written in expressions.
2. Curried function can be applied to one argument only, giving another function that may be useful in its own right

Definitions

- There are definitions for different kinds of values:
  - Definitions of fixed values.

```
pi :: Float
pi = 3.14159
```

  - Some definitions of functions use *conditional expressions*

```
smaller :: (Integer, Integer) → Integer
smaller(x,y) = if x ≤ y then x else y
```

Definitions

- The same expressions can be defined using *guarded equations.*

```
smaller :: (Integer, Integer) → Integer
smaller(x,y)
      | x ≤ y = x
      | X > y = y
```

  - Each clause consists of a condition, or guard, and an expression, which is separated from the guard by an = sign.
  - The main advantage of guarded expressions is when there are three or more clauses in a definition.