Functional languages

Intro

What makes a language functional? The fact that "*functions are first class citizen of the language*". This means that functions can be passed as arguments to other functions, returned as values from other functions, and assigned to variables or data structures.

Some basics

Analyzing the `C` grammar we can identify **expressions** and **statemets**.

- **Statements** are single line of code of type `statement`
- **Expressions** are entities that can be evaluated and have a type

Let's describe the `C` grammar in BNF notation:

```
expr ::= constant            constant ::= 0 | 1 | 2 | 3 | ...
  |  (expr)                            |  'a' | 'b' | ...
  |  x                                 |  1.0 | 2.45 | ...
  |  expr++                            |  "string..."
  |  ++expr
  |  expr--                  block ::= statement ;
  |  --expr                         |  statement ; block
  |  expr = expr
  |  expr += expr            cases :: = case constant: block
  |  expr -= expr                  |   case constant: block cases
  |  ...
  |  expr + expr             statement ::= return expr
  |  expr - expr                       |  if (expr) { block }
  |  expr * expr                       |  if (expr) { block } else { block }
  |  expr / expr                       |  for (statement ; expr ; expr ) { block }
  |  expr % expr                       |  while (expr) { block }
  |  expr << expr                      |  do { block } while (expr)
  |  expr >> expr                      |  switch (expr) { cases }
  |  expr & expr                       |  break
  |  expr | expr                       |  continue
  |  expr ^ expr                       |  goto x
  |  ~ expr
  |  - expr
  |  f(expr1, ..., exprN)
  |  & expr
  |  expr ? expr : expr
  |  * expr
  |  expr[expr]
  |  ! expr
  |  expr && expr
  |  expr || expr
  |  expr == expr
  |  expr < expr
  |  expr <= expr
  |  expr > expr
  |  expr >= expr
  |  expr.x
```

Reduction

Reduction is the evaluation of an expression. With this process every part of the expression is "reduced" to a smaller form until we get to a *ground value*. Let's take $1 + 2 * 4$ as example:

$$\overbrace{\overbrace{\underbracket{\underbracket{1+2}_{3} * 4}_{12}}^{\text{int}}}^{\text{int}}$$

On the lower brackets we can see the result of the evaluation of the arithmetical operations. From the upper brackets instead we can see that the reduction preserves the types of the expression.