|  |  |
| --- | --- |
| Гидра | Haskell |
| (Chars.isLower c) | isLower c (по идее с может быть только Int) Нужен импорт в Haskell: import Data.Char, чтобы функция распознавалась |
| (Chars.isUpper c) | isUpper c +import Data.Char |
| (Chars.toLower c) | toLower c +import Data.Char |
| (Chars.toUpper c) | toUpper c  +import Data.Char |
| (Math.neg x) | negate x |
| (Math.add x y)  (Math.add (Math.add x y) z) | x + y любое количество переменных приемлемо  x+y+z |
| (Math.sub x 1)  (Math.sub x y) | x – 1  x-y |
| (Math.mul x 1)  (Math.mul x y) | x \* 1  x\*y |
| (Math.div x 1) | x / 1 (целочисленное деление) |
| (Math.mod x 1) | x `mod`1 |
| (Math.rem x 1) | x `rem` 1 |
| (Logic.and (x) (y)) | x&&y |
| (Logic.not (x)) | not(x) |
| (Logic.or (x) (y)) | x || y |
| (Strings.cat str) | concat str |
| (Strings.cat2 str1 str2) | Str1 ++ str2 |
| (Strings.fromList codes) | map chr codes  +import Data.Char |
| (Strings.isEmpty str) | null str |
| (Strings.length s) | length s |
| (Strings.toList s) | map ord codes  +import Data.Char |
| (Strings.toLower s) | map toLower s +import Data.Char |
| (Strings.toUpper s) | map toUpper s  +import Data.Char |
| (Logic.ifElse (x) True False) | if x then True else False Можно в несколько строк |
| (Equality.equalInt32 x y) | x == y |
| (Equality.gtInt32 x y) | Если будут обычные типы, то x>y |
| (Equality.gteInt32 x y) | Если будут обычные типы, то x>=y |
| (Equality.ltInt32 x y) | Если будут обычные типы, то x<y |
| (Equality.lteInt32 x y) | Если будут обычные типы, то x<=y |
| Равенство стрингов | X==y |
| (Logic.not (Equality.equalInt32 x y)) можно не генерировать отдельно, так как составная функция, но, если хочешь, можно и искать отдельно | x/=y ИЛИ not(x==y) |
| (Lists.at i xs) | xs !! i |
|  |  |
| (Lists.concat2 xs ys) | ++ |
| (Lists.cons x xs) | x : xs |
| (Lists.filter p xs) | Filter p xs |
| (Lists.head xs) | head xs |
| (Lists.last xs) | last xs |
| (Lists.length xs) | Length xs |
| (Lists.reverse xs) | reverse xs |
| (Lists.tail xs) | tail xs |
|  |  |

Вложенность реализована. Любое количество переменных

ПРО ЛИСТЫ!!!   
Так как в гидре функции с листами выглядят так:

head :: [a] -> a

head = L.head  
a- параметризованный тип  
  
когда я мы задаём вместо а Int/String И тд, гидра всё равно преобразует всё это в неопределенный тип, который выглядит как head\_ :: ([t0] -> t0).   
  
то есть при обратном преобразовании t0 это a

ещё странно выводятся массивы в сгенерированном коде в Haskell

aaaaa :: [String]

aaaaa = [

"123"]