Idris — type classes and equality

Daria Walukiewicz-Chrząszcz

7 marca 2017

Solutions of exercises from Lab1

- Zad1.idr
- Zad2.idr
- Zad3.idr
- Zad4.idr
- Zad5.idr

Type classes

- similar to type classes in Haskell
- there can be many implementations for one type

(see Eq.idr Tree.idr)

Equality in Idris

- == is not adequate
- equality defined at the level of types

(see EqNat.idr, ExactLength.idr)

Dependent types and dependent pattern-matching

- types depend on values
- no syntactical differences between type and value
- types are values
- type of the result of a function can depend on values of arguments

Dependent types and dependent pattern-matching

- types depend on values
- no syntactical differences between type and value
- types are values
- type of the result of a function can depend on values of arguments

Dependent types and dependent pattern-matching

- types depend on values
- no syntactical differences between type and value
- types are values
- type of the result of a function can depend on values of arguments

Dependent types and dependent pattern-matching

- types depend on values
- no syntactical differences between type and value
- types are values
- type of the result of a function can depend on values of arguments

Totality checking

Function is *total* if it

- covers all possible inputs
- is well-founded (in recursive calls arguments are decreasing)
- does not use any data types which are not strictly positive
- does not call any non-total functions

Totality checking

Function is *total* if it

- covers all possible inputs
- is well-founded (in recursive calls arguments are decreasing)
- does not use any data types which are not strictly positive
- does not call any non-total functions

Totality checking

Function is *total* if it

- covers all possible inputs
- is well-founded (in recursive calls arguments are decreasing)
- does not use any data types which are not strictly positive
- does not call any non-total functions

Totality checking

Function is *total* if it

- covers all possible inputs
- is well-founded (in recursive calls arguments are decreasing)
- does not use any data types which are not strictly positive
- does not call any non-total functions

Totality checking

Function is *total* if it

- covers all possible inputs
- is well-founded (in recursive calls arguments are decreasing)
- does not use any data types which are not strictly positive
- does not call any non-total functions