1 Small examples

For these examples, assume that there is a length indexed vector,

$n : \mathbb{N}, A : \star \vdash Vec\,A\,n$

and functions defined with the types

$zip : (A : \star) \to (B : \star) \to (n : \mathbb{N}) \to Vec\,A\,n \to Vec\,B\,n \to Vec\,(A, B)\,n$

$rep : (n : \mathbb{N}) \to Vec\,\mathbb{N}\,n$

$head : (A : \star) \to (n : \mathbb{N}) \to Vec\,A\,(suc\,n) \to A$

$head' : (A : \star) \to (n : \mathbb{N}) \to (n > 0) \to Vec\,A\,n \to A$

$append : (A : \star) \to (x\,y : \mathbb{N}) \to Vec\,A\,x \to Vec\,A\,y \to Vec\,A\,(x + y)$

1.1 Definitions hidden in another module

1.1.1 Definitional Equality

If

$f : \mathbb{N} \to \mathbb{N}$ and

$f' : \mathbb{N} \to \mathbb{N}$

have the exact same definition but are defined in different modules such that their implementation is hidden, then I am aware of no dependent type system that will type the application

$g = \lambda x.zip\,\mathbb{N}\,\mathbb{N}\,(f\,x)\,(rep\,(f\,x))(rep\,(f'\,x))\ :\ (x : \mathbb{N}) \to Vec\,(\mathbb{N}, \mathbb{N})\,(f\,x)$

Since the application typing relies on definitional equality, the only conventional option is to change the structure of the code.

My proposed system would insert a check automatically:

- If g is ever called on an $\mathbb{N}$ that could differentiate f and f' an error message can be given with the concrete inequality, giving the programmer actionable evidence to fix the bug.
- The check will be automatically exercised with automated tests, such that if there is ever a difference it will be found.

1.1.2 Propositional property

If

$more : \mathbb{N} \to \mathbb{N}$

such that the output is always greater then one, but defined in a different module such that the implementation is hidden and the module does not expose a proof of the property.

Then there is no straightforward way to define the function

$g = \lambda x.head'\,\mathbb{N}\ (more\,x)\ ?\ (rep\ (more\,x))\ :\ \mathbb{N} \to \mathbb{N}$

The conventional ways of trying to write that function will:

- use an explicit test, and change the output type to reflect the possibility of failure
- carefully postulate the behavior is correct after a manual check has failed so that the program will only get stuck when $more\,x$ is not greater then 0

1.1.3 Current solutions

Agda avoids this issue by not allowing modules to hide definitions. I believe Idris' modules hide implementations and discourages this kind of dependent use.

2 Vector Associativity

This example is often used to justify Heterogeneous equality. Prove:

$(A : \star) \to (x\,y\,z : \mathbb{N}) \to (xx : Vec\,A\,x) \to (yy : Vec\,A\,y) \to (zz : Vec\,A\,z) \to Id\ (append\,xx\ (append\,yy\,zz))\ (append\ (append\,xx\,yy)\ zz)$

Unfortunately, this is not even a problem that can be posed in many type systems with the most standard Id type, since (x+(y+z)) is usually not definitionally equal to ((x+y)+z).

With some effort it would it is possible to construct the proof in a different flavor of equality, assuming it is possible to establish $Id\,(x+(y+z))\,((x+y)+z)$.

- In my system, the associativity of addition will be presumed, and runtime checks will insure there is never an observable violation
- Additionally automated tests will perform a search for a violation of the implied specification of $(x + (y + z)) = ((x + y) + z)$