Homework 2
Type Isomorphisms

98-317: Hype for Types

Due: 31 Jan 2022 at 8:00 PM

Introduction

This week we learned about type isomorphisms. In this homework, you will use Standard ML to write proofs of some type isomorphisms.

Turning in the Homework: Submit your `handin.zip` file to the "Algebraic Data Types" assignment on Gradescope.

Representing Isomorphism Proofs as SML Values

Recall that two types τ_1 and τ_2 are isomorphic (which we write as $\tau_1 \simeq \tau_2$) if there exist functions $f : \tau_1 \to \tau_2$ and $g : \tau_2 \to \tau_1$ such that $f \circ g = \mathsf{id}_{\tau_2}$ and $g \circ f = \mathsf{id}_{\tau_1}$ (where id_τ represents the identity function for type τ). In this spirit, we define the SML type

```
type ('a, 'b) isomorphic = ('a -> 'b) * ('b -> 'a)
```

with the intent that a value of type (τ_1, τ_2) `isomorphic` represents a proof of the theorem $\tau_1 \simeq \tau_2$. It's worth noting that while SML's type system will automatically check that the functions have the correct type, it will *not* check that their compositions are identity functions – instead, our autograder will check this.

SML/NJ has product types and sum types built in. A type $\tau_1 \times \tau_2$ is represented as τ_1 `*` τ_2 and has values of the form `(`e_1`,` e_2`)`. A type $\tau_1 + \tau_2$ is represented as `(`τ_1`,` τ_2`) either`, and has values of the form `Left` e_1 and `Right` e_2.

We've also provided a type inhabited by no values, named `void`:[1]

```
datatype void = Void of void
```

In the following tasks you will be asked to prove isomorphisms of types by writing SML values.

Example Prove

$$\forall \alpha, \beta.\ \alpha * \beta \simeq \beta * \alpha$$

by implementing a value `commutativity_of_product` of type

```
('a * 'b, 'b * 'a) isomorphic
```

Solution:

```
val commutativity_of_product = (
  fn (x,y) => (y,x),
  fn (y,x) => (x,y)
)
```

[1] SML's syntax doesn't allow declaring a datatype with no constructors, so this recursive type is a hacky way to ensure that no values of this type can be created.

Your Task Implement the type isomorphisms in a structure `Isomorphisms : ISOMORPHISMS`. The signature for `ISOMORPHISMS` is in the file `isomorphisms.sig`.

For Your Consideration (Extra)

- Define a value of type `(nat, (nat, nat) either) isomorphism` to prove that $|\mathbb{N}| + |\mathbb{N}| = |\mathbb{N}|$.
- Define:

  ```
  datatype 'a tree = Empty | Node of 'a tree * 'a * 'a tree
  datatype 'a rose = Rose of 'a * 'a rose list
  ```

  Then, define a value of type `('a tree, 'a rose) isomorphism` to prove that `'a tree` and `'a rose` are isomorphic.[2]

[2] https://en.wikipedia.org/wiki/Left-child_right-sibling_binary_tree