Homework 3
Type Isomorphisms

98-317: Hype for Types

Due: 4 February 2020 at 6:30 PM

Introduction

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

This homework is divided into two parts: "Fun" and "Optional" (also fun). You will receive full credit for this homework if you turn in something for the "Fun" portion.

Turning in the Homework: Email your hw3.sml file to `autograder@averycowan.com` as an attachment. I have it set up to forward to my gmail. Or you can print it out and bring it to class.

Representing Isomorphism Proofs as SML Values

You may have heard the phrase "types are theorems; programs are proofs". We're going to use that philosophy to have you turn in proofs which we can autograde.

Recall that two types τ_1 and τ_2 are isomorphic (which we write as $\tau_1 \cong \tau_2$) 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 $\tau_1 \cong \tau_2$. 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 `INL` e_1 and `INR` e_2[1].

We've also provided a type inhabited by no values, named `void`:[2]

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

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

Example Task Prove

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

by implementing a value `commutativity_of_product` of type

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

Solution:

```
local
  fun f (x, y) = (y, x)

  fun g (y, x) = (x, y)
in
  val commutativity_of_product
    : ('a * 'b, 'b * 'a) isomorphic
    = (f, g)
end
```

[1] The `either` type constructor and `INL` and `INR` constructors are found in the `Either` module, which we have opened at the top of your code file.

[2] 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.

Fun

Fun Task 1 Prove

$$\forall \alpha, \beta.\ \alpha + \beta \cong \beta + \alpha$$

by implementing a value `commutativity_of_sum` of type

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

Fun Task 2 Prove

$$\forall \alpha.\ 1 \times \alpha \cong \alpha$$

by implementing a value `identity_of_product` of type

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

Fun Task 3 Prove

$$\forall \alpha.\ 0 + \alpha \cong \alpha$$

by implementing a value `identity_of_sum` of type

```
((void, 'a) either, 'a) isomorphic
```

Fun Task 4 Prove

$$\forall \alpha, \beta, \gamma.\ (\alpha \times \beta) \times \gamma \cong \alpha \times (\beta \times \gamma)$$

by implementing a value `associativity_of_product` of type

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

Fun Task 5 Prove

$$\forall \alpha, \beta, \gamma.\ (\alpha + \beta) + \gamma \cong \alpha + (\beta + \gamma)$$

by implementing a value `associativity_of_sum` of type

```
((('a, 'b) either, 'c) either, ('a, ('b, 'c) either) either) isomorphic
```

Fun Task 6 Prove

$$\forall \alpha, \beta, \gamma.\ \alpha \times (\beta + \gamma) \cong (\alpha \times \beta) + (\alpha \times \gamma)$$

by implementing a value `distributivity` of type

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

Optional: Arrows as Exponents

We haven't yet talked about how arrow types fit into the algebraic interpretation of types. One way to think of them is as exponents, where $\tau_1 \to \tau_2$ corresponds to $\tau_2^{\tau_1}$.

Optional Task 1 Prove

$$\forall \alpha.\ \alpha^1 \cong \alpha$$

by implementing a value `one_exponent` of type

```
(unit -> 'a, 'a) isomorphic
```

Optional Task 2 Prove

$$\forall \alpha.\ 1^\alpha \cong 1$$

by implementing a value `one_to_power` of type

```
('a -> unit, unit) isomorphic
```

Optional Task 3 Prove

$$\forall \alpha.\ \alpha^{1+1} \cong \alpha \times \alpha$$

by implementing a value `two_exponent` of type

```
((unit, unit) either -> 'a, 'a * 'a) isomorphic
```