Why does `Applicative (StateT s f)` instance require `Monad f` bound? #134

New issue

Open **ktt3ja** opened this issue on May 3, 2015 · 1 comment

ktt3ja commented on May 3, 2015

The instance implementation for Applicative (StateT s f) only needs me to use the `pure` function. Even the doc states "given a `Applicative f`". Yet, if I were to change the bound `Monad f` to `Applicative f`, I'd get the following error:

```
src/Course/StateT.hs:66:10:
    Could not deduce (Bind f)
      arising from the superclasses of an instance declaration
    from the context (Applicative f)
      bound by the instance declaration at src/Course/StateT.hs:66:10-50
    Possible fix:
      add (Bind f) to the context of the instance declaration
    In the instance declaration for `Applicative (StateT s f)'
```

How does this error come about?

thejohnfreeman commented on Nov 14, 2015

Can you offer an implementation of `<*>` for `StateT s f` that only uses `pure` and `<*>` for `f`? I'm not sure it's possible, because the function within a `StateT` is essentially monadic. Consider the signature of `<*>`:

```
StateT g <*> StateT h = ...
-- g :: s -> f (a -> b, s)
-- h :: s -> f (a, s)
-- g and h have signatures like that of the left argument of >>=
```

We need to apply `g` and `h` to get the function and argument respectively. We cannot apply them to the same state, though, or we will have to drop one of the two intermediate states, since the state type `s` is not a monoid:

```
-- choose the first intermediate state
StateT g <*> StateT h = StateT $ \s -> keepFirst <$> g s <*> h s
  where keepFirst (f, s') (x, _) = (f x, s')
-- choose the second intermediate state
StateT g <*> StateT h = StateT $ \s -> keepSecond <$> g s <*> h s
  where keepSecond (f, _) (x, s') = (f x, s')
```

The intermediate state is locked within an applicative in the result of `g s`, and we want to thread it through `h`. That sequencing suggests to me that we need `f` to be a monad, and in fact, the `Applicative` instance for `StateT` in the transformers package requires `f` to be a monad, and its implementation of `<*>` uses the monadic function `ap`.

I tried but could not succeed without using it as a monad. Here's the implementation I settled on:

Labels
None yet

Milestone
No milestone

Assignee
No one assigned

Notifications
Subscribe
You're not receiving notifications from this thread.

2 participants

```
StateT g <*> StateT h = StateT $ \s -> do
  (f, s') <- g s
  (x, s'') <- h s'
  return (f x, s'')
```

Write Preview

Leave a comment

Attach files by dragging & dropping or selecting them.

Styling with Markdown is supported

Comment