

Chapter 1

Proofs of Parser Laws

The approach taken to prove the following parser laws for `parsley` is via equational reasoning on `gigaparsec` semantics, under the assumption that their semantics are equivalent. While there is no formal proof of this equivalence at the present, `gigaparsec` was designed to have semantics equivalent to `parsley`'s.

1.1 Left absorption for `fmap`

```

f <$> empty
=   { applicative functor law }
pure f <*> empty
=   { definition of <*> }
liftA2 ($) (pure f) empty
=   { semantics of liftA2 }
Parsec $ λst ok err →
  let ok' x st' = (unParsec empty) st' (ok . (x $)) err
  in (unParsec $ pure f) st ok' err
=   { semantics of empty }
Parsec $ λst ok err →
  let ok' x st' = (unParsec $ raise (`emptyErr` 0)) st' (ok . (x $)) err
  in (unParsec $ pure f) st ok' err
=   { semantics of raise }
Parsec $ λst ok err →
  let ok' x st' = (unParsec $ Parsec $ λst'' _ bad →
    useHints bad (emptyErr st'' 0) st') st' (ok . (x $)) err
  in (unParsec $ pure f) st ok' err
=   { β-reduction }
Parsec $ λst ok err →
  let ok' x st' = useHints err (emptyErr st' 0) st'
  in (unParsec $ pure f) st ok' err
=   { semantics of pure }
Parsec $ λst ok err →
  let ok' x st' = useHints err (emptyErr st' 0) st'
  in (unParsec $ Parsec $ λst'' ok'' _ → ok'' f st'') st ok' err
=   { β-reduction }
Parsec $ λst ok err →
  let ok' x st' = useHints err (emptyErr st' 0) st'

```

```
      in ok' f st
=    { inline ok' }
      Parsec $ \st ok err → useHints err (emptyErr st 0) st
=    { rearrange and  $\alpha$ -conversion }
      Parsec $ \st _ bad → useHints bad ((`emptyErr` 0) st) st
=    { fold definition of raise }
      raise (`emptyErr` 0)
=    { fold definition of empty }
      empty
```