ST 모나드와 다형성

rank-n type & impredicative type

문순원

ST 모나드의 필요

가변자료

- 많은 상황에서 효율성을 위해 가변자료형이 필요
- 하스켈에서는 가변상태 제어에 IO 모나드를 사용

ST 모나드의 필요

가변자료

- 많은 상황에서 효율성을 위해 가변자료형이 필요
- 하스켈에서는 가변상태 제어에 IO 모나드를 사용

IO 모나드의 문제점

- IO 모나드는 가변자료 이상의 기능을 제공
- IO 모나드의 연산결과는 IO 타입을 탈출할수 없다
- IO 모나드를 강제로 탈출할경우 참조투명성이 훼손

```
unsafePerformIO :: IO a -> a
```

ST 모나드의 개선점

The ST monad allows for destructive updates, but is escapable (unlike IO)

— base 라이브러리 문서

ST 모나드의 개선점

> *The ST monad allows for destructive updates, but is escapable (unlike IO)*
>
> — base 라이브러리 문서

- ST 모나드는 결정론적이다
- ST 모나드는 탈출할수 있으면서도 참조투명성을 해치지 않는다
- **순수 함수 인터페이스를 가진 명령형 코드**

어떻게 가능한가?

```haskell
runST :: (forall s. ST s a) -> a
```

어떻게 가능한가?

```haskell
runST :: (forall s. ST s a) -> a
```

- ST 모나드를 안전하게 탈출
- **rank-2 다형성**

다형함수를 인자로 받기

```haskell
foo :: ??
foo f = (f 1, f True)

bar :: (Int, Bool)
bar = foo id
```

다형함수를 인자로 받기

```haskell
foo :: ??
foo f = (f 1, f True)

bar :: (Int, Bool)
bar = foo (id :: forall a. a -> a)
```

다형함수를 인자로 받기

```haskell
foo :: (forall a. a -> a) -> (Int, Bool)
foo f = (f 1, f True)

bar :: (Int, Bool)
bar = foo (id :: forall a. a -> a)
```

다형함수를 인자로 받기

```haskell
foo :: (forall a. a -> a) -> (Int, Bool)
foo f = (f 1, f True)

bar :: (Int, Bool)
bar = foo (id :: forall a. a -> a)
```

- rank-2 타입
- **표준 하스켈에서는 허용되지 않음**

rank에 따른 타입의 분류

*The **rank** of a type describes the depth at which universal quantifiers appear contravariantly*

—Practical type inference for arbitrary-rank types

rank에 따른 타입의 분류

> *The **rank** of a type describes the depth at which universal quantifiers appear contravariantly*
>
> —*Practical type inference for arbitrary-rank types*

rank-0

- `Int -> Int`

rank에 따른 타입의 분류

> *The **rank** of a type describes the depth at which universal quantifiers appear contravariantly*
>
> *—Practical type inference for arbitrary-rank types*

rank-0

- `Int -> Int`

rank-1

- `forall a. Int -> a -> a`
- `Int -> (forall a. a -> a)`

rank에 따른 타입의 분류

> *The **rank** of a type describes the depth at which universal quantifiers appear contravariantly*
>
> *—Practical type inference for arbitrary-rank types*

rank-0

- `Int -> Int`

rank-1

- `forall a. Int -> a -> a`
- `Int -> (forall a. a -> a)`

rank-2

- `(forall a. a -> a) -> Int`

rank에 따른 타입의 분류

> *The **rank** of a type describes the depth at which universal quantifiers appear contravariantly*
>
> *—Practical type inference for arbitrary-rank types*

rank-0

- `Int -> Int`

rank-1

- `forall a. Int -> a -> a`
- `Int -> (forall a. a -> a)`

rank-2

- `(forall a. a -> a) -> Int`

rank-3

- `((forall a. a -> a) -> Int) -> Int`

rank에 따른 타입의 분류

> *The **rank** of a type describes the depth at which universal quantifiers appear contravariantly*
>
> —*Practical type inference for arbitrary-rank types*

rank-0

- `Int -> Int`

rank-1

- `forall a. Int -> a -> a`
- `Int -> (forall a. a -> a)`

rank-2

- `(forall a. a -> a) -> Int`

rank-3

- `((forall a. a -> a) -> Int) -> Int`

rank에 따른 타입시스템의 분류

- rank-1 다형성 : rank가 1 이하인 타입을 허용
- rank-2 다형성 : rank가 2 이하인 타입을 허용
- rank-3 다형성 : rank가 3 이하인 타입을 허용

  ...
- rank-k 다형성 : rank가 k 이하인 타입을 허용

- rank-n (higher-rank) 다형성 : 임의의 rank를 가진 타입을 허용

rank에 따른 타입시스템의 분류

- rank-1 다형성 : rank가 1 이하인 타입을 허용
- rank-2 다형성 : rank가 2 이하인 타입을 허용
- rank-3 다형성 : rank가 3 이하인 타입을 허용

  ...
- rank-k 다형성 : rank가 k 이하인 타입을 허용

- rank-n (higher-rank) 다형성 : 임의의 rank를 가진 타입을 허용

타입추론이 결정가능

GHC의 언어 확장 RankNTypes

RankNTypes

- 하스켈에 rank-n 다형성 지원을 추가

GHC의 언어 확장 RankNTypes

RankNTypes

- 하스켈에 rank-n 다형성 지원을 추가
- 이제는 가능하다

```haskell
foo :: (forall a. a -> a) -> (Int, Bool)
foo f = (f 1, f True)
```

GHC의 언어 확장 RankNTypes

RankNTypes

- 하스켈에 rank-n 다형성 지원을 추가
- 이제는 가능하다

```haskell
foo :: (forall a. a -> a) -> (Int, Bool)
foo f = (f 1, f True)
```

Rank2Types

- 이론적으로 타입추론이 결정가능
- 실제로 GHC에 결정가능한 rank-2 타입추론이 구현된적은 없음

GHC의 언어 확장 RankNTypes

RankNTypes

- 하스켈에 rank-n 다형성 지원을 추가
- 이제는 가능하다

```haskell
foo :: (forall a. a -> a) -> (Int, Bool)
foo f = (f 1, f True)
```

Rank2Types

- 이론적으로 타입추론이 결정가능
- 실제로 GHC에 결정가능한 rank-2 타입추론이 구현된적은 없음
- RankNTypes의 별칭
- 쓰지마세요

rank vs kind

- rank ≠ kind

rank vs kind

- rank ≠ kind

```
Int -> Int                          :: Type (rank-0)
forall a. Int -> a -> a             :: Type (rank-1)
Int -> (forall a. a -> a)           :: Type (rank-1)
(forall a. a -> a) -> Int           :: Type (rank-2)
```

rank vs kind

- rank ≠ kind

```
Int -> Int                          :: Type (rank-0)
forall a. Int -> a -> a             :: Type (rank-1)
Int -> (forall a. a -> a)           :: Type (rank-1)
(forall a. a -> a) -> Int           :: Type (rank-2)
```

- rank에 상관없이 kind ‘Type’을 가진다

rank vs kind

- rank ≠ kind

```
Int -> Int                          :: Type (rank-0)
forall a. Int -> a -> a             :: Type (rank-1)
Int -> (forall a. a -> a)           :: Type (rank-1)
(forall a. a -> a) -> Int           :: Type (rank-2)
```

- rank에 상관없이 kind 'Type'을 가진다
- 하스켈에서는 rank에 따른 차별이 존재

```
[not] :: [Bool -> Bool]      -- okay
[id]  :: [forall a. a -> a] -- not okay
```

rank vs kind

- rank ≠ kind

```
Int -> Int                          :: Type (rank-0)
forall a. Int -> a -> a             :: Type (rank-1)
Int -> (forall a. a -> a)           :: Type (rank-1)
(forall a. a -> a) -> Int           :: Type (rank-2)
```

- rank에 상관없이 kind 'Type'을 가진다
- 하스켈에서는 rank에 따른 차별이 존재

```
[not] :: [Bool -> Bool]      -- okay
[id]  :: [forall a. a -> a] -- not okay
```

polymorphic types are themselves not first class

—*A Quick Look at Impredicativity*

Impredicativity

```haskell
id :: forall p. p -> p
mono :: [Bool] -> [Bool]
poly :: forall a. [a] -> [a]


mono_a :: [Bool] -> [Bool]
mono_a = id @([Bool] -> [Bool]) mono

poly_a :: forall b. [b] -> [b]
poly_a @b = id @([b] -> [b]) (poly @b)

poly_b :: forall b. [b] -> [b]
poly_b = id @(forall a. [a] -> [a]) poly
```

Impredicativity

```
id :: forall p. p -> p
mono :: [Bool] -> [Bool]
poly :: forall a. [a] -> [a]


mono_a :: [Bool] -> [Bool]
mono_a = id @([Bool] -> [Bool]) mono

p ~ [Bool] -> [Bool]
```

Impredicativity

```haskell
id :: forall p. p -> p
mono :: [Bool] -> [Bool]
poly :: forall a. [a] -> [a]


poly_a :: forall b. [b] -> [b]
poly_a @b = id @([b] -> [b]) (poly @b)

p ~ [b] -> [b]
a ~ b
```

Impredicativity

```
id :: forall p. p -> p
mono :: [Bool] -> [Bool]
poly :: forall a. [a] -> [a]


poly_b :: forall b. [b] -> [b]
poly_b = id @(forall a. [a] -> [a]) poly

p ~ forall a. [a] -> [a]
```

Impredicativity

```haskell
id :: forall p. p -> p
mono :: [Bool] -> [Bool]
poly :: forall a. [a] -> [a]


poly_b :: forall b. [b] -> [b]
poly_b = id @(forall a. [a] -> [a]) poly
```

```
p ~ forall a. [a] -> [a]
```

- 타입변수를 다형타입으로 인스턴스화

Impredicativity

```
id :: forall p. p -> p
mono :: [Bool] -> [Bool]
poly :: forall a. [a] -> [a]


poly_b :: forall b. [b] -> [b]
poly_b = id @(forall a. [a] -> [a]) poly

p ~ forall a. [a] -> [a]
```

- 타입변수를 다형타입으로 인스턴스화
- Impredicative 다형성
- rank-n 다형성의 일반화
  `[id] :: [forall a. a -> a]`
- GHC는 '아직' impredicativity를 제대로 지원하지 않음

ST & runST

```
action :: ST s a
runST :: forall a. (forall s. ST s a) -> a
```

> *A computation of type `ST s a` returns a value of type `a`, and execute in "thread" `s`.*
> *`runST` return the value computed by a state thread. The `forall` ensures that the internal state used by the `ST` computation is inaccessible to the rest of the program.*
>
> — base 라이브러리 문서

ST & runST

```
action :: ST s a
runST :: forall a. (forall s. ST s a) -> a
```

> *A computation of type ST s a returns a value of type a, and execute in "thread" s.*
> *runST return the value computed by a state thread. The forall ensures that the internal state used by the ST computation is inaccessible to the rest of the program.*
>
> — base 라이브러리 문서

- ST 모나드의 연산결과를 ST 모나드에서 탈출시킴
- 내부 상태가 ST 모나드를 탈출하지 못하게 함

ST 모나드의 내부 상태

내부 상태 ≈ 참조 타입

- `STRef s a`
- `MVector s a`
- `HashTable s k a`

ST 모나드의 내부 상태

내부 상태 ≈ 참조 타입

- `STRef s a`
- `MVector s a`
- `HashTable s k a`
- 스레드 s에 종속된 a 타입의 가변 상태를 가리키는 참조
- ST 모나드 안에서 값을 읽고 쓸수 있음

```haskell
newSTRef :: a -> ST s (STRef s a)
readSTRef :: STRef s a -> ST s a
writeSTRef :: STRef s a -> a -> ST s ()
```

runST의 참조투명성 보장

```haskell
runST :: forall a. (forall s. ST s a) -> a

escapeInt :: Int
escapeInt = runST action where
  action :: forall s. ST s Int
  action = do
    ref <- newSTRef 0
    readSTRef ref
```

```
forall s. ST s a ~ forall s. ST s Int
∴ a ~ Int
```

- Int는 ST 모나드를 탈출할 수 있음

runST의 참조투명성 보장

```haskell
runST :: forall a. (forall s. ST s a) -> a

escapeSTRef :: STRef s Int..?
escapeSTRef = runST action where
  action :: forall s. ST s (STRef s Int)
  action = do
    ref <- newSTRef 0
    return ref
```

`forall s. ST s a ~ forall s. ST s (STRef s Int)`

`∴ a ~ STRef s Int`

Couldn't match type 'a' with 'STRef s Bool' because type variable 's' would escape its scope

- 내부상태 `STRef s Int`는 ST 모나드를 탈출할 수 없음

ST 모나드와 벡터

```
freeze       :: MVector s a -> ST s (Vector a)
unsafeFreeze :: MVector s a -> ST s (Vector a)

thaw       :: Vector a -> ST s (MVector s a)
unsafeThaw :: Vector a -> ST s (MVector s a)

runST  :: (forall s. ST s a) -> a
create :: (forall s. ST s (MVector s a)) -> Vector a
```

- 불변벡터-가변벡터간 변환

ST 모나드와 벡터

```
freeze       :: MVector s a -> ST s (Vector a)
unsafeFreeze :: MVector s a -> ST s (Vector a)

thaw       :: Vector a -> ST s (MVector s a)
unsafeThaw :: Vector a -> ST s (MVector s a)

runST  :: (forall s. ST s a) -> a
create :: (forall s. ST s (MVector s a)) -> Vector a
```

- 불변벡터-가변벡터간 변환
- 복사를 하지 않는 구현은 참조 투명성을 깨뜨림
- create는 안전하면서도 복사가 없음

create 사용 예시 (병합 정렬)

https://gist.github.com/damhiya/d46e7197b5186795f2fd3ae49eade029

```haskell
sortM :: Ord a => MVector s a -> ST s ()
sortM u = ..

sort :: Ord a => Vector a -> Vector a
sort v =
  if V.length v <= 1 then
    v
  else runST $ do
    v' <- V.thaw v
    sortM v'
    V.freeze v'
```

create 사용 예시 (병합 정렬)

https://gist.github.com/damhiya/d46e7197b5186795f2fd3ae49eade029

```haskell
sortM :: Ord a => MVector s a -> ST s ()
sortM u = ..

sort :: Ord a => Vector a -> Vector a
sort v =
  if V.length v <= 1 then
    v
  else runST $ do
    v' <- V.thaw v
    sortM v'
    V.unsafeFreeze v'
```

create 사용 예시 (병합 정렬)

https://gist.github.com/damhiya/d46e7197b5186795f2fd3ae49eade029

```haskell
sortM :: Ord a => MVector s a -> ST s ()
sortM u = ..

sort :: Ord a => Vector a -> Vector a
sort v =
  if V.length v <= 1 then
    v
  else V.create $ do
    v' <- V.thaw v
    sortM v'
    return v'
```

- 불필요한 복사 제거
- 참조 투명성이 항상 보장됨

Q&A