타입 추론

- 함수의 타입 추론 규칙 (논리 증명의 Modus Ponen)

$$\frac{\texttt{f :: a -> b} \qquad \texttt{m :: a}}{\texttt{(f m) :: b}}$$

- Monomorphic

  $+_{Int}$: Int -> Int -> Int

  $length_{Char}$:[Char] -> Int

Proof Tree에 의한 타입 유도

ord ‘c’ $+_{Int}$ 3_{Int} = ($+_{Int}$) (ord ‘c’) (3_{Int})

ord :: Char -> Int ‘c’ :: Char

(ord ‘c’)::Int 3_{Int}::Int $+_{Int}$:: Int ->(Int ->Int)

($+_{Int}$) (ord ‘c’) :: Int -> Int 3_{Int}::Int

(($+_{Int}$) (ord ‘c’)) (3_{Int}) :: Int

(ord ‘c’) $+_{Int}$ False 은 타입 트리를 구성할 수 없으므로 에러

Polymorphic 타입 체킹

- 타입 변수(type variables)
- monotype : 타입 변수를 사용하지 않는 경우

```
length :: [a] -> Int
```

- 문맥에 따른 타입 변수의 구체화 (instantiation)

```
f (x, y) = (x, ['a' .. y])
```

y 는 Char 타입이 되어야 함.

```
f :: (a, Char) -> (a, [Char])
```

- `g (m, zs) = m + length zs`
  `+` 가 두 `Int` 를 가질 경우, `length :: [b] -> Int`
  `g :: (Int, [b]) -> Int`

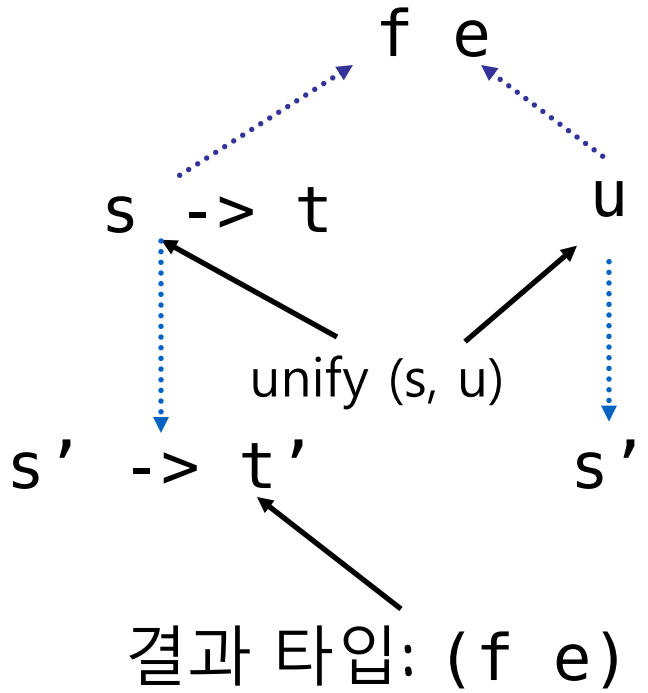
- `h = g . f` (함수의 합성)
  ```
  f :: (a, Char) -> (a, [Char])
  g :: (Int, [b]) -> Int
  ```
  `(a, [Char])` 와 `(Int, [b])`가 unified 되어야 함.

- Most General Common Instance : `(Int, [Char])`.
  `h :: (Int, Char) -> Int`

- Unification 예
  - 대상 : (a, [a]) 와 ([b], c)
  - 결과 : ([b], [[b]])

- 타입([b], [[b]])의 instance 예
  ([Bool], [[Bool]]), ([[c]], [[[c]]])

Polymorphic 타입 체킹



타입 추론 : foldr 함수

- foldr 함수

```
foldr f s []          = s
foldr f s (x:xs)      = f x (foldr f s xs)
```

- 추론 과정
  - 일단 foldr 함수의 타입을 다음과 같이 가정

    ```
    foldr :: (a -> b -> c) -> d -> [e] -> f
    ```

  - f oldr의 결과 타입: 두번째 인수인 d와 f 의 결과 타입이 동일 : unify(d,f)
  - f 의 두번째 인수 타입 : unify(b,f)
  - f 의 첫번째 인수 타입 : unify(a,e)
  - f 의 결과 값의 타입 : unify(f, c)

```
foldr :: (a -> b -> b) -> b -> [a] -> b
```

Algebraic Type

- 일반적 형태

```
data Typename
        =  Con1 t11 ... t1k1 |
           Con2 t21 ... t2k2 |
             . . .           |
           Conn tn1 ... tnkn |
```

- 재귀적 정의

```
data Expr = Lit Int | Add Expr Expr | Sub Expr Expr
```

타입 클래스에 의한 Overloading

- Overloading
  - 타입에 따라 동일한 이름의 함수를 여러 번 정의하는 경우

- 타입 클래스와 Instance

```
class Eq a where                    // Java : interface Eq
  (==) :: a -> a -> Bool            // Java의 abstract 메소드에 해당됨

instance Eq Temp where              // Java : Temp implements Eq
  Cold == Cold   = True             // (==) 메소드 구현
  Hot == Hot     = True
```

Eq의 예

- 이진 트리 정의

```
data Btree = Nil | Node Int Btree Btree
t1 = Node 10 (Node 20 Nil Nil) (Node 30 Nil Nil)
t2 = Node (2+8) (Node (10+10) Nil Nil) (Node 30 Nil Nil)
```

- Btree에 대한 Eq 정의

```
instance Eq Btree where
  Nil == Nil                       = True
  Node x t1 t2 == Node y t3 t4 = (x == y) && (t1 == t3) && (t2 == t4)
  _ == _                           = False
```

- 자동 생성

```
data Btree = Nil | Node Int Btree Btree deriving (Eq)
```