Típuslevezetés példákon keresztül

Hanák D. Péter
2005. febr. 20.

Magasabbrendű polimorf függvények típusa

1. `fun f1 p q r s = p(q r s)`

   Az `f1` *részlegesen alkalmazható* függvényként van definiálva.

   Az `f1`-et a `p` argumentumra alkalmazva olyan függvényt kapunk, amelyet a `q` argumentumra alkalmazva olyan függvényt kapunk, amelyet az `r` argumentumra alkalmazva olyan függvényt kapunk, amelyet az `s` argumentumra alkalmazva kapjuk meg a függvényalkalmazás eredményét.

   `f1` típusának meghatározását a definíció jobb oldalának elemzésével kezdjük. `p` függvény típusú érték, mert *függvényalkalmazási pozícióban* áll. A `p(q r s)` függvényalkalmazás kiértékelését az SML a zárójelben álló `q r s` argumentum kiértékelésével kezdi (vö. mohó kiértékelés). A zárójelen belül a `q` is függvényalkalmazási pozícióban van, ezért `q` is függvény-típusú érték. A `q`-t az `r` argumentumra alkalmazva olyan függvényt kapunk, amelyet az `s` argumentumra alkalmazva kapjuk az eredményt. `r` és `s` típusáról nem tudunk közelebbit, ezért a típusokat egy-egy típusváltozóval jelöljük.

   Típusváltozóként a szokásos módon alfát `('a)`, bétát `('b)`, gammát `('c)`, deltát `('d)` használunk. A fentiek szerint:

`p : 'a -> 'b`	* `p`-nek *egyetlen* argumentuma van: `(q r s)`
`r : 'c`	* `r` típusáról nem tudunk semmi közelebbit
`s : 'd`	* `s` típusáról sem tudunk semmi közelebbit
`q : 'c -> 'd -> 'a`	* `q` eredménye a `p` argumentuma

   Már csak össze kell rakni az egyes részeket:

   ```
   f1 : ('a -> 'b) -> ('c -> 'd -> 'a) -> 'c -> 'd -> 'b
   ```

   Megjegyzések: `('a -> 'b)`-t és `('c -> 'd -> 'a)`-t zárójelbe kell rakni, hiszen az összes `->` típusoperátor azonos precedenciájú. `f1` és `p` alkalmazásának eredménye azonos típusú (a példában `'b`-val jelöltük).

2. `fun uncurry f (x, y) = f x y`

   `uncurry` is részlegesen alkalmazható függvényként van definiálva: első argumentumát `f`-fel jelöljük; második argumentuma egy pár, `(x, y)`-nal jelöljük. Milyen érték `f`? A választ a definíció jobb oldalán találjuk: mivel `f` függvényalkalmazási pozícióban áll, függvénynek, mégpedig részlegesen alkalmazható függvénynek kell lennie. Ezzel:

   ```
   x : 'a
   y : 'b
   f : 'a -> 'b -> 'c
   (x, y) : 'a * 'b
   uncurry : ('a -> 'b -> 'c) -> 'a * 'b -> 'c
   ```

Megjegyzés: `'a * 'b` köré nem kell zárójelet rakni, mert a `*` típusoperátor precedenciája nagyobb, mint a `->` típusoperátoré.

3. `val mapmap = map map`

   A `map` alapfüggvény, szignatúráját, típusát tudni illik:

   ```
   val map f ls :  ('a -> 'b) -> 'a list -> 'b list =
                   ('a -> 'b) -> '(a list -> 'b list)
   ```

   A két típuskifejezés ekvivalens, hiszen a `->` típusoperátor jobbra köt.

   A `mapmap` definíciójában a `map`-et saját magára alkalmazzuk részlegesen; eredményül egyargumentumú függvényt kapunk.

   Ha a `map`-et egyargumentumú függvénynek tekintjük, az argumentumának `'a -> 'b`, az eredményének pedig `'a list -> 'b list` a típusa.

   A `mapmap` típusát úgy kapjuk meg, hogy a `map` eredményének típusát leíró kifejezésben az `'a` típusváltozót `('a -> 'b)`-val, a `'b` típusváltozót pedig `('a list -> 'b list)`-tel helyettesítjük:

   ```
   mapmap : ('a -> 'b) list -> ('a list -> 'b list) list
   ```

4. `fun swap f x y = f y x`

   Mind a `swap`, mind `f`-fel jelölt első argumentuma részlegesen alkalmazható függvény (az utóbbi a definíció jobb oldalából látható). `swap`-nak és `f`-nek azonos típusú eredményt kell adnia.

   ```
   y : 'a
   x : 'b
   f : 'a -> 'b -> 'c
   swap : ('a -> 'b -> 'c) -> 'b -> 'a -> 'c
   ```

5. ```
   fun fold f e [] = e
     | fold f e (x::xs) = fold f (f e x) xs
   ```

   A `fold` fenti változata részlegesen alkalmazható függvény. `f`-fel jelölt első argumentuma ugyancsak részlegesen alkalmazható függvény, ez a definíció második sorában a jobb oldalon álló `(f e x)` kifejezésből látszik. `e`-vel jelölt második argumentuma tetszőleges típusú érték, `[]`-vel, ill. `(x::xs)`-sel jelölt harmadik argumentuma pedig tetszőleges típusú értékek listája. `fold` rekurzív hívásából látható, hogy `f` eredményének `e`-vel azonos típusúnak kell lennie.

   ```
   e : 'a
   x : 'b
   f : 'a -> 'b -> 'a
   fold : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a
   ```
   ```