

POLIMORFIZMUS, EXPANZIÓ



Paraméteres polimorfizmus

- Az identitásfüggvény és típusa: `fun id x = x, id : 'a -> 'a.`
Az `mosml` válasza: `val 'a id = fn : 'a -> 'a.`
Az `id` *politípusú* név, az `'a` az `id` típusparamétere.
- Az `=` és a `<>` műveletet *készen kapjuk* a legtöbb típusra.
A típusuk: `=, <> : ''a * ''a -> bool.` A `''` *egyenlőségi típust* jelöl: az ilyen típusú értékeken az egyenlőségvizsgálat elvégezhető.
- Az egyenlőségvizsgálat *korlátozottan* polimorf: nem minden értékre végezhető el. Pl. egy f és egy g függvény akkor és csak akkor egyenlő, ha $\forall x \cdot fx = gx$. Ezt *általánosságban* lehetetlen eldönteni.
- Mi a `<`, `>`, `<=`, `>=` típusa?
Pl. az `op<=`-re az `mosml` válasza: `val it = fn : int * int -> bool.`
E négy művelet *ad-hoc* módon polimorf, a nevek *többszörösen terhelhetők*, alapértelmezés szerint `int` típusú értékekre alkalmazhatók.
- Nézzünk néhány példát (lásd a következő lapokon is)!
Az `=` részlegesen alkalmazható változata legyen: `fun eq x y = x = y.`
A típusa: `eq : ''a -> ''a -> bool.`

Példák az eq használatára ('a eq : 'a -> 'a -> bool)

| A kifejezés | Az mosml válasza |
|--------------------------------------|---|
| <code>eq 3 3;</code> | <code>> val it = true : bool</code> |
| <code>eq "id" "idn";</code> | <code>> val it = false : bool</code> |
| <code>eq id id;</code> | <pre>! Toplevel input: ! eq id id; ! ^^ ! Type clash: expression of type ! 'a -> 'a ! cannot have equality type ''b</pre> |
| <code>eq 3;</code> | <code>> val it = fn : int -> bool</code> |
| <code>eq "id";</code> | <code>> val it = fn : string -> bool</code> |
| <code>val eqStr_id = eq "id";</code> | <code>> val eqStr_id = fn : string -> bool</code> |

- Az `id` függvény, típusa (`'a -> 'a`) nem egyenlőségi típus!
- Az `eq "id"` függvényértéket ad eredményül, ezért az `eqStr_id` függvényt jelöl. Olyan függvényt, amely az `"id"` füzérre alkalmazva `true`, minden más esetben `false` értéket ad eredményül.

Példák az `id` használatára (`'a id : 'a -> 'a`)

A kifejezés

Az mosml válasza

| | |
|----------------------------------|---|
| <code>id 3;</code> | <code>> val it = 3 : int</code> |
| <code>id "id";</code> | <code>> val it = "id" : string</code> |
| <code>id round;</code> | <code>> val it = fn : real -> int</code> |
| <code>id id;</code> | <code>! Warning: Value polymorphism:</code> <code>! Free type variable(s) at top level</code> <code> in value identifier it</code> |
| | <code>> val it = fn : 'a -> 'a</code> |
| <code>id id 6.9;</code> | <code>> val it = 6.9 : real</code> |
| <code>fn x => id id x;</code> | <code>> val 'b it = fn : 'b -> 'b</code> |

Érték-polimorfizmus

- Az SML ún. *érték-polimorfizmust* használ.
 - Az SML a típusváltozókat, ahol csak tudja, általánosítja (vö. $\text{fn } x \Rightarrow \text{id id } x$).
 - Az `mosml` a nem általánosítható típusváltozókat *szabad típusváltozóként* kezeli (vö. id id).
- Tekintsük a `val x = e` deklarációt.
 - Az SML az x típusában előforduló szabad típusváltozókat akkor általánosítja, ha e ún. *nem-expanzív* kifejezés.
 - Ez csupán *szintaktikai* követelmény: egy kifejezés *nem-expanzív*, ha megfelel a *nexp* szintaktikai kategóriát leíró nyelvtani szabályoknak (ld. a következő lapokon).

Nem-expanzív kifejezés (egyszerűsítve)

- Nem-expanzív kifejezés (*nexp*: non-expansive expression)

| | | |
|-------------------------|---|--|
| <i>nexp ::= scon</i> | különleges állandó | special constant |
| <i>longvid</i> | (esetleg minősített) értéknév | (possibly qualified) value identifier |
| { < <i>nexprow</i> > } | nem-expanzív elemekből álló rekord | record of non-expansive expressions |
| (<i>nexp</i>) | nem-expanzív kifejezés zárójelben | parenthesized non-expansive expression |
| <i>nexp</i> : <i>ty</i> | nem-expanzív kifejezés típusmegkötéssel | typed non-expansive expression |
| <i>fn match</i> | függvénykifejezés | function expression |

- Nem-expanzív kifejezéssor (*nexprow*: non-expansive expression row)

nexprow ::= lab = nexp <, nexprow>

Példák nem-expanzív és expanzív kifejezésekre

- Egy nem-expanzív kifejezés egyszerűen: érték (azaz tovább nem egyszerűsíthető, ún. *kanonikus* kifejezés).

```
val x = length;
> val 'a x = fn : 'a list -> int
```

`length` egy név, ezért nem-expanzív. Az `x` típusát leíró `'a list -> int` típuskifejezésben az `'a` szabad típusváltozó általánosítható, ezt tükrözi a definíció bal oldalán az `'a x`.

- Az `(fn f => f) length` kifejezés értéke is `length`, de expanzív, mert nem vezethető le a fenti nyelvtani szabályok alapján.

```
val x = (fn f => f) length;
! Warning: Value polymorphism:
! Free type variable(s) at top level in value identifier x
> val x = fn : 'a list -> int
```

Példák nem-expanzív és expanzív kifejezésekre (folyt.)

- Expanzív kifejezés esetén az SML nem általánosítja a típusváltozót (ld. az előző példában 'a-t).
- Az `mosml` 'a-t meghagyja szabad típusváltozónak; és csak az *x első alkalmazásakor* köti le.

```
x ["abc", "def"];
! Warning: the free type variable 'a has been instantiated
~~~~~to string
> val it = 2 : int
x;
> val it = fn : string list -> int
```

- Ha az 'a-t egyszer már lekötöttük, más típushoz nem köthető; az `x` nem politípusú név.

```
x [123, 456, 789];
! Toplevel input:
!   x [123, 456, 789];
!     ^^^
! Type clash: expression of type
!   int
! cannot have type
!   string
```


η -expanszió

- A típusváltó általánosítása mindig kikényszeríthető a deklaráció jobb oldalának ún. η -expansziójával (olvasd: eta-expansziójával).
- Az η -expanszió az e kifejezést a nem-expanzív $\text{fn } y \Rightarrow e \ y$ kifejezéssel helyettesíti.
- Példa:

```

• val x1 = fn y => ((fn f => f) length) y;
  > val 'b x1 = fn : 'b list -> int

```

- A deklarációban a külső zárójelpár el is hagyható:

```
val x1 = fn y => (fn f => f) length y;
```

- Az $x1$, mint látható, politípusú név, ezért:

```

x1 ["abc", "def"];
> val it = 2 : int
x1 [123, 456, 789];
> val it = 3 : int

```