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
`eq 3 3;`	`> val it = true : bool`
`eq "id" "idn";`	`> val it = false : bool`

`eq id id;`	`! Toplevel input:`
	`! eq id id;`
	`!    ^^`
	`! Type clash: expression of type`
	`!   'a -> 'a`
	`! cannot have equality type ''b`

`eq 3;`	`> val it = fn : int -> bool`
`eq "id";`	`> val it = fn : string -> bool`
`val eqStr_id = eq "id";`	`> val eqStr_id = fn : string -> bool`

- 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
`id 3;`	`> val it = 3 : int`
`id "id";`	`> val it = "id" : string`
`id round;`	`> val it = fn : real -> int`

`id id;`	`! Warning: Value polymorphism:`
	`! Free type variable(s) at top level`
	`in value identifier it`
	`> val it = fn : 'a -> 'a`

`id id 6.9;`	`> val it = 6.9 : real`
`fn x => id id x;`	`> val 'b it = fn : 'b -> 'b`

Érték-polimorfizmus

- Az SML ún. *érték-polimorfizmust* használ.
  - Az SML a típusváltozókat, ahol csak tudja, általánosítja (vö. `fn x => 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)

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

- A típusváltozó általánosítása mindig kikényszeríthető a deklaráció jobb oldalának ún. *η-expanziójával* (olvasd: eta-expanziójával).
- Az η-expanzió az `e` kifejezést a nem-expanzív `fn y => 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
    ```
    ```