

FUNKCIONÁLIS PROGRAMOZÁS

A funkcionális programozás néhány jellemzője

Funkcionális, más néven applikatív programozás

- Funkcionális = függvényalapú, függvényközpontú
- Applikatív = (függvényeket argumentumokra) alkalmazó
- Függvényjel, más néven operátor
- Argumentum, más néven paraméter vagy operandus
- Kifejezés, kiértékelés (más néven egyszerűsítés, redukció), érték

Fő különbségek az imperatív és a funkcionális programozás között

- Változó: frissíthető, ill. nem frissíthető (vö. értékadás, ill. kötés)
- Ismétlés: ciklus, ill. rekurzió (vö. helyesség bizonyítása teljes indukcióval)
- Függvények: függvényeljárás, ill. „tisztá” függvény (vö. mellékhatás, ill. mellékhatás-mentesség)
- Állapotváltozások sorozata, ill. időtlenség, emlékezet nélkülség

Az SML néhány jellemzője

A Standard Meta Language (SML) néhány jellemzője

- Rekurzív típus: lineáris (lista) és nemlineáris (pl. fa)
- Magasabb rendű függvény (argumentuma és/vagy eredménye függvény)
- „Erősen típusos” (*strong typing*; ellenőrzés fordításkor)
- Típuslevezetés (*type derivation*)
- Polimorfizmus (többszörös terheléses és paraméteres)
- Modularitás (szignatúra, struktúra, funktor)
- Absztrakt típus

SML-megvalósítások

- Moscow ML (mosml): <http://www.dina.kvl.dk/~sestoft/mosml.html>
- Poly/ML: debugging! <http://www.polymml.org>
- Standard ML of New Jersey (sml): <http://cm.bell-labs.com/cm/cs/what/smlnj>

TÍPUS ÉS FÜGGVÉNY

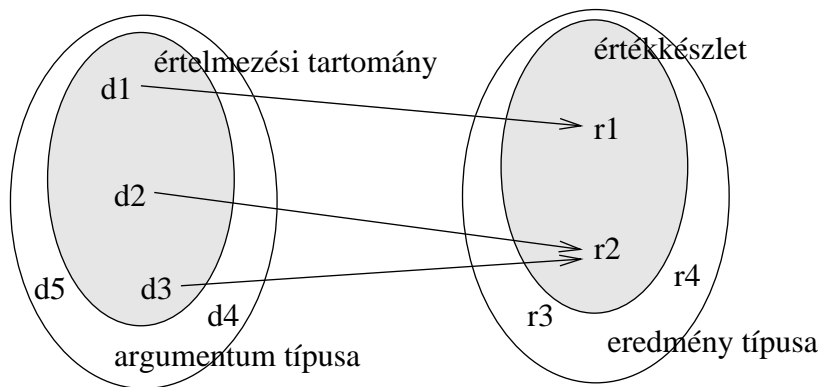
A típus és a függvény fogalma

- A típus fogalma
 - ◇ A típus értékek egy halmaza (pl. egész típus ~ az egész számok egy (rész)halmaza)
 - ◇ Jelölése az ún. *típuselméletben*: $\alpha, \beta \dots$
 - ◇ Típusállandó (azaz konkrét típus) jelölése az SML-ben: `int`, `real`, `char`, `string`...
 - ◇ Típusváltozó jelölése az SML-ben $\alpha, \beta \dots$ helyett: `'a`, `'b`...
- A függvény fogalma
 - ◇ A függvény valamely D halmaznak valamely R halmazba való olyan *egyértelmű* leképzése, amelyet a (d, r) rendezett párok halmaza ad meg, ahol $d \in D$ és $r \in R$.
 - ◇ A d a függvény argumentuma (paramétere), az r az eredménye
 - ◇ A D a függvény értelmezési tartománya, az R az értékkészlete
 - ◇ A típusos nyelvekben d is, r is *meghatározott* típusú
 - ◇ A függvény értelmezési tartománya \subseteq az argumentum típusa
 - ◇ A függvény értékkészlete \subseteq az eredmény típusa

A függvény mint érték

- A függvény „teljes jogú” (*first-class*) érték a funkcionális programozási nyelvekben
 - ◇ A függvény típusa általában: $\alpha \rightarrow \beta$, ahol az α az argumentum, a β az eredmény típusát jelöli
 - ◇ A függvény maga is: érték. *Függvényérték*.
 - ◇ Fontos: a függvényérték *nem* a függvény egy *alkalmazásának* az eredménye!
 - ◇ Példák függvényértékre
 - * `sin` (a típusa: *valós* \rightarrow *valós*)
 - * `round` (a típusa: *valós* \rightarrow *egész*)
 - * `o` (függvénykompozíció; a típusa: $((\beta \rightarrow \gamma) * (\alpha \rightarrow \beta)) \rightarrow (\alpha \rightarrow \gamma)$)
 - ◇ Példák függvényalkalmazásra
 - * `round 5.4 = 5`, azaz egy *egész* típusú érték az eredménye ennek a függvényalkalmazásnak
 - * `round o sin` (a típusa: *valós* \rightarrow *egész*)
 - * `(round o sin)1.0 = 1` (a típusa: *egész*)

A függvény mint leképzés



Függvények osztályozása, ill. alkalmazása

- Függvények osztályozása
 - ◇ Parciális függvény: értelmezési tartomány \subset argumentum típusa (figyelem: hibák forrása lehet!)
 - ◇ Teljes függvény: értelmezési tartomány = argumentum típusa
 - ◇ Szürjektív függvény: értékkészlet = eredmény típusa
 - ◇ Nem-szürjektív függvény: értékkészlet \subset eredmény típusa
 - ◇ Injektív függvény: a leképzés kölcsönösen egyértelmű
 - ◇ Az $f : \alpha \rightarrow \beta$ injektív függvény inverze: $f^{-1} : \beta \rightarrow \alpha$
 - ◇ Bijektív = injektív + szürjektív, azaz f bijektív, ha f^{-1} teljes függvény
- Függvények alkalmazása
 - ◇ *Függvényalkalmazást* jelöl az f és e jelek egymás mellé írása („*juxtapozicionálása*”): $f e$ azt jelenti, hogy f -et alkalmazzuk e -re.
 - ◇ Általánosabban: az $f e$ kifejezésben az e tetszőleges olyan kifejezés, amelynek az értéke az f értelmezési tartományába esik.
 - ◇ Még általánosabban: az $f e$ kifejezésben az f függvényértéket adó tetszőleges kifejezés, e pedig tetszőleges olyan kifejezés, amelynek értéke az f értelmezési tartományába esik.

Két- vagy többargumentumú függvények

- Függvény alkalmazása két- vagy több argumentumra
 1. Az argumentumokat *összetett adatnak* – párnak, rekordnak, listának stb. – tekintjük
 - ◇ pl. $f(1, 2)$ az f függvény alkalmazását jelenti az $(1, 2)$ párra,
 - ◇ pl. $f[1, 2, 3]$ az f függvény alkalmazását jelenti az $[1, 2, 3]$ listára.
 2. A függvényt több egymás utáni lépésben alkalmazzuk az argumentumokra, pl. $f\ 1\ 2 \equiv (f\ 1)\ 2$ azt jelenti, hogy
 - ◇ az első lépésben az f függvény alkalmazzuk az 1 értékre, ami egy függvényt ad eredményül,
 - ◇ a második lépésben az első lépésben kapott függvényt alkalmazzuk a 2 értékre, így kapjuk meg az $f\ 1\ 2$ függvényalkalmazás (vég)eredményét.
- Az $f\ 1\ 2$ esetben az f függvényt *részlegesen alkalmazható* függvénynek nevezzük.
- A programozó szabadon dönthet, hogy a függvényt részlegesen alkalmazható vagy pl. egy párra alkalmazható formában írja meg. A különbség a szintaxisban van. A részlegesen alkalmazható változat, mint látni fogjuk, rugalmasabban használható.
- Infix jelölés: $x \oplus y \equiv a \oplus$ függvény alkalmazása az (x, y) párra mint argumentumra

Függvények alkalmazása az SML-ben

- Az SML-ben az f és az e tetszőleges *név* lehet, amelyeket megfelelően *szeparálni* kell egymástól: $f\ e$, vagy $f(e)$, vagy $(f)\ e$
- Szeparátor: nulla, egy vagy több *formázó* karakter (\lfloor , $\backslash t$, $\backslash n$ stb.). Nulla db formázó karakter elegendő pl. $a\ ($ előtt és $)$ után.
- A szeparátor a legerősebb balra kötő infix operátor az SML-ben.
- Példák:


```
Math.sin 1.00 (Math.cos)Math.pi round(3.17)
2 + 3          (real) (3 + 2 * 5)
```
- Függvények egy csoportosítása az SML-ben
 - ◇ Beépített függvények, pl. $+$, $*$ (mindkettő infix), $real$, $round$ (mindkettő prefix)
 - ◇ Könyvtári függvények, pl. $Math.sin$, $Math.cos$, $Math.pi$ (0 argumentumú!)
 - ◇ Felhasználó által definiálható függvények, pl. $terület$, $/\backslash$, $head$

FÜGGVÉNYEK AZ SML-BEN

SML-példa: Egyszeres Hamming-távolságú ciklikus kód

- A függvényt *táblázattal* adjuk meg:

00	01	fn	00 =>	01
01	11		01 =>	11
11	10		11 =>	10
10	00		10 =>	00
- Változatok („klózek”): minden lehetséges esetre egy változat.
- Az fn (olvasd: *lambda*), névtelen függvényt, *függvénykifejezést* vezet be.
- A függvény néhány alkalmazása:
 - ◇ $(fn\ 00\ =>\ 01\ | 01\ =>\ 11\ | 11\ =>\ 10\ | 10\ =>\ 00)\ 10$
 - ◇ $(fn\ 00\ =>\ 01\ | 01\ =>\ 11\ | 11\ =>\ 10\ | 10\ =>\ 00)\ 11$
 - ◇ $(fn\ 00\ =>\ 01\ | 01\ =>\ 11\ | 11\ =>\ 10\ | 10\ =>\ 00)\ 111$
- Mintaillesztés, egyesítés
- Érthető, de nem robusztus (vö. parciális a függvény!).

SML-példa: modulo n alapú inkrementálás

- A függvényt *algoritmussal* adjuk meg (nem táblázattal)
 - ◊ n nem lehetne változó,
 - ◊ túl sok változatot kellene felírni stb.
- `fn i => (i + 1) mod n`
 - ◊ az i ún. kötött változó, a névtelen függvény argumentuma
 - ◊ az n ebben a kifejezésben szabad változó, és nincs értéke (!)
 - ◊ az n -et is le kell kötni mint a függvény argumentumát
- `fn n => fn i => (i + 1) mod n`
- A függvény néhány alkalmazása:
 - ◊ `(fn n => (fn i => (i + 1) mod n)) 128 111`
 - ◊ `(fn n => (fn i => (i + 1) mod n)) 4 ~7`
 - ◊ `(fn n => (fn i => (i + 1) mod n)) 128 6.0` – hiba!

Értékdeklaráció SML-ben: függvényérték deklarálása

- Név kötése függvényértékhez
 - ◊ `val incMod = fn n => fn i => (i + 1) mod n`
 - ◊ `val kovKod = fn 00 => 01 | 01 => 11 | 11 => 10 | 10 => 00`
- Szintaktikus édesítőszerrel
 - ◊ `fun incMod n i = (i + 1) mod n`
 - ◊ `fun kovKod 00 = 01`
 | `kovKod 01 = 11`
 | `kovKod 11 = 10`
 | `kovKod 10 = 00`
- Alkalmazásuk argumentumra
 - ◊ `incMod 128 111`
 - ◊ `kovKod 01`

Fejtkomment

Írjunk *fejtkommentet* minden (függvény)érték-deklarációhoz!

- (* `incMod n i = (i+1) modulo n` szerint
 PRE: `n > i >= 0`
 *)
`fun incMod n i = (i+1) mod n`
- (* `kovKod cc = az egyszeres Hamming-távolságú, kétbites,`
 ciklikus kódkészlet `cc`-t követő eleme
 PRE: `cc in 00, 01, 11, 10`
 *)
`fun kovKod 00 = 01`
 | `kovKod 01 = 11`
 | `kovKod 11 = 10`
 | `kovKod 10 = 00`

LISTÁK

Lista: definíciók, konstruktorok

- Definíciók

1. A *lista* azonos típusú elemek véges (de nem korlátos!) sorozata.
2. A lista olyan *rekurzív* lineáris adatszerkezet, amely azonos típusú elemekből áll, és
 - ◇ vagy üres,
 - ◇ vagy egy elemből és az elemet követő listából áll.

- Konstruktorok

- ◇ Az üres lista jele a *nil* *konstruktorállandó*. *nil* típusa 'a list.
- ◇ A *::* *konstruktoroperátor* új listát hoz létre egy elemből és egy (esetleg üres) listából (infix, 5-ös precedenciájú, jobbra köt, típusa 'a * 'a list -> 'a list).
- ◇ A *nil* helyett általában a *[]* jelet használjuk (szintaktikus édesítőszer).
- ◇ A *::*-ot négyespontnak vagy *cons*-nak olvassuk (vö. *constructor*, ami a függvény hagyományos neve a λ -kalkulusban és egyes funkcionális nyelvekben).

Lista: jelölések, minták

- Példák

- ◇ Lista létrehozása konstruktorokkal

```
[ ]          nil          #" " :: nil
3 :: 5 :: 9 :: nil    = 3 :: (5 :: (9 :: nil))
```

- ◇ Szintaktikus édesítőszer lista jelölésére

```
[3, 5, 9]          = 3 :: 5 :: 9 :: nil
```

- ◇ Vigyázat! A Prolog listajelölése hasonló, de vannak lényeges különbségek:

SML	Prolog		SML	Prolog	
[]	[]	azonos	(x::xs)	[X Xs]	különböző
[1,2,3]	[1,2,3]	azonos	(x::y::ys)	[X,Y Ys]	különböző

- Minták

A *[]*, *nil* konstruktorállandóval és a *::* konstruktoroperátorral felépített kifejezések, valamint a *[x1, x2, ..., xn]* listajelölés mintában is alkalmazhatók.