FP-GYIK

Gyakori kérdések – válaszokkal

*

a funkcionális programozás témaköréből

Hanák D. Péter

2005. május 4.

Tartalomjegyzék

Bevezető

Az FP-GYIK[1] a *Deklaratív programozás* tárgy hallgatóinak kérdéseiből és nemritkán válaszaiból készült összeállítás, amelyet a tárgy `dp-l@iit.bme.hu` levelezési listájára 2000. elejétől 2004. végéig érkező levelekből válogattam ki. Ha tehát valaki a saját stílusára ismerne, az nem a véletlen műve. Sok-sok hallgatónak tartozom köszönettel a jó kérdésekért és a még jobb válaszokért, közülük többen is mindkét térfelen küzdöttek az idők során (ugye, rablóból lesz . . .).

A teljesség igénye nélkül álljon itt jónéhány név a közreműködők közül, köztük olyanoké is, akik a tárgynak „hivatalosan” talán soha nem voltak hallgatói:

> Benkő Tamás, Lukácsy Gergely, Varga Szilvia, Sója Katalin, Wagner Ambrus, Tarján Péter, Pallinger Péter, Altrichter Márta, Pálfalvi Tamás, Hanák Dávid, Berki Lukács Tamás, Békés András György, Ottucsák György, Drótos Márton, Szeredi Tamás, Patai Gergely, Kiss Zoltán, Kovács Zoltán, Csík László, Dóra László, Szabó Péter.

Az eredeti „hangszerelést”, ahol nem ment az érthetőség rovására, sok helyen megőriztem. Sokszor csaknem ugyanaz a kérdés és válasz többféle változatban is olvasható; ha többször kérdezték ugyanazt, valószínűleg nehezebben érthető dologról van szó, érdemes hát többször foglalkozni vele.

Az FP-GYIK első néhány verziójában biztosan lesznek hibák, elírások: az észrevételeket köszönettel várom a „hanak at inf dot bme dot hu” címre.

Göd, 2005. február — Hanák Péter

[1] GYIK = Gyakori kérdések – válaszokkal

1. fejezet

Az MOSML-értelmező használata

1.1. Kérdés. Az SML-t hogy kéne használni? Prologban eligazodom, mert azt láthattuk az előadáson, hogy hogy kell használni. De az SML-t... Meg se bírok benne moccanni.

1.1. Válasz. ***Hallgató:*** Ez szerintem is jó kérdés... Én letöltöttem a tárgy honlapjáról egy html-es segitséget, ami anno az eredeti mosml-honlapról származik. Azzal se vittem sokra... Odáig jutottam, hogy be tudtam loadolni, nem jelzett hibát loadoláskor, de utána meg sem moccant. Csak CONTROL+Z-re reagált. Szal nem sok... (Amúgy egy fájlból akartam betölteni a dolgot.... A honlapon van róla valami kis leírás.)

A kérdező folytatja: Végül arra sikerült rájönnöm, hogy az mosml.exe az indítható.

Hallgató: Mert nem olvastad el a `readme.txt-t`.:)))) Ott mindent leír. Még be kell írni két sort az `autoexec.bat`-ba is...

A kérdező folytatja: Na de hogy kéne abba vmi programot betölteni (consultálni, mint a Prologban)??? Vagy ott kell mindent bepötyögni? Nem hinném...

Hallgató: Nem, be is lehet tölteni egy `use` függvénnyel, bár nekem nem igazán ment...

A kérdező folytatja: És miért csak akkor szól vissza, ha ";"-t is leütök?? A pontosvessző az utasítás vége? vagy mi? Lehet, hogy én vagyok béna, de nekem ez a ";" teljesen új... Vagy nem figyeltem eléggé az előadáson??

Hallgató: Hát akkor már ketten nem figyeltünk...

A kérdező folytatja: Más. Mi az az emacs editor? Azzal kéne használni valahogy az SML-t? Mert ha igen, honnan lehet megszerezni?

Hallgató: Tudomásom szerint linux alá szövegszerkesztőnek indult, de kicsit már kinötte ezt a mivoltát. Azért még szöveget is lehet vele szerkeszteni. :))) Egy változata letölthető a tárgy honlapjáról: <http://dp.iit.bme.hu/download.html>, legfrissebb változata pedig az eredeti Gnu emacs honlapról: <http://www.gnu.org/software/emacs>.

A kérdező folytatja: Más: a weben merre lehet letölteni mosml kézikönyvet, dokumentációt, használati útmutatót????

Hallgató: A tárgy honlapjáról: <http://dp.iit.bme.hu/download.html> vagy az eredeti mosml honlapról: <http://www.dina.kvl.dk/~sestoft/mosml.html>.

Másik hallgató: Azt hiszem, nekem is hasonló érzéseim vannak a dologgal kapcsolatban, annak ellenére, hogy már valamennyire tudom használni.

Szóval az egyik fontos „meta-függvény” a `use`, pl. `use "filenev.sml"`. Ez hasonlóan működik a Sicstus `consult`-jához, már amennyire abból a három próbálkozásból meg tudtam ítélni... Például:

```
Moscow ML version 1.44 (August 1999)
Enter ‘quit();’ to quit.
- use "imax.sml";
[opening file "imax.sml"]
> val imax = fn : int * int -> int
[closing file "imax.sml"]
> val it = () : unit
- imax(2, 3);
> val it = 3 : int
```

ahol az `imax.sml`:

```
(* imax (a, b) = a és b maximuma *)
fun imax (a, b) = if a >= b then a else b
```

Másik hallgató: Ha interaktív módban írsz be valamit, akkor csak a `;` után kezd el foglalkozni a beírt dolgokkal, tehát a sorvége önmagában még nem jelent semmit (persze a `;` után is kell egy sorvége a pufferelés miatt). Az sml fájlban viszont nem kell, mert ott a fájlvége jelre „indul be” az értelmező.

1.2. Kérdés. Hogyan kell az mosml-be betölteni egy `vmi.sml` kiterjesztésű fájlt???

1.2. Válasz. ***Hallgató:*** Nagyon egyszerű: `use "vmi.sml"`.

1.3. Kérdés. Hogy lehet sml-ben (mosml, polyml) betölteni 1 .sml kiterjesztésű fájlt?

1.3. Válasz. ***Oktató:*** Több helyen le van írva, hogy az interaktív környezetben a `use` függvényt kell használni (pl. `use "fuggvenykem.sml"`), az órákon időnként elmondtam. Részletesen lásd pl. a következő forrásokat (mindkettőt egyébként is érdemes megnézni):

a) Module Meta, <http://dp.iit.bme.hu/mosml/doc/mosmllib.pdf>-ben.

b) The interactive system: mosml, <http://dp.iit.bme.hu/mosml/doc/manual.pdf>

Mindkettő az mosml-disztribúció része, ezért valószínűleg megtalálja a saját gépén is...:)

1.4. Kérdés. SML-ben nem tudom sehogysem include-olni a `list` library-t. Tudna valaki segíteni?

1.4. Válasz. ***Hallgató:*** Nekem valami `load "List"` (nagy kezdőbetűvel!) rémlik, de lehet, hogy nem jól emlékszem.

1.5. Kérdés. A következő problémám van mosml-ben. Pl. `Math.sin` használatakor a következőt írja ki:

```
> - Math.sin 1.0;
> ! Toplevel input:
> ! Math.sin 1.0;
> ! ^^^^^^^^
> ! Cannot access unit Math before it has been loaded.
```

Viszont a `List` könyvtár függvényeit tudom használni. Mi lehet a probléma?

1.5. Válasz. ***Oktató:*** A `load "Math";` függvényalkalmazással előbb be kell tölteni az mosml-interpreterbe. A `List` könyvtárat az mosml az indulásakor mindig betölti.

1.6. Kérdés. A kisháziban használom a `Math.sqrt` függvényt. Hogyan lehet rábírni az mosml-t, hogy betöltse a `Math` modult, amikor valaki betölti a kisházit `use "khf-ml1.sml"`-lel?

1.6. Válasz. ***Hallgató:*** Interaktív módban a `load "Math"`-t vagy a `loadOne "Math"`-t használd.

Ha a programodat `use`-zal töltöd be, akkor akár a fájlban is lehet a `load "Math"`, de kiadhatod kézzel a `use` előtt is. A `load` ti. ***csak*** interaktív módban működik, fordításkor nem, akkor viszont hibát okoz. Ez utóbbi miatt (mivel a házi feladatokat mosmlc-vel lefordítjuk) a beküldött programból ki kell kommentezni vagy ki kell törölni a `load`-okat.

Az `open` más célra való: pl. az `open Math` hatására elhagyhatók a `Math.` prefixek a függvényhívások elől. Inkább ne használd, jobb, egyértelműbb a prefix kiírása, még ha hosszabb lesz is a kód.

1.7. Kérdés. Röviden fogalmazva: van valamilyen debugger-szerűség az mosml-ben?

1.7. Válasz. ***Oktató:*** Röviden fogalmazva: debugger nincs.

1.8. Kérdés. Már volt egy hasonló kérdés a Prologról, de sml-ben hogyan lehet megoldani, hogy a teljes listát írja ki, ne harapja le a végét?

1.8. Válasz. ***Oktató:*** Hosszú lista, ill. egymásba skatulyázott adatszerkezetek esetén `printVal` (és maga az SML-értelmező is) alapesetben csak az első 200 listaelemet, ill. legfeljebb 20 szintet ír ki. A hosszat a `printLength`, a szintek számát a `printDepth` frissíthető változó szabályozza. Mindkét érték felülírható.

1.1. Windows

1.9. Kérdés. A windowsos verziót töltöttem le, és a futtatáshoz be kellene írni az `autoecex.bat`-ba a következőt:

```
set PATH=C:\dos;  ...  ;C:\mosml\bin
set MOSMLLIB=C:\mosml\lib
```

Viszont én win2000-et használok, és abban nincs `autoexec.bat`, :(így nem is működik a progi.

1.9. Válasz. ***Hallgató:*** Win2k alatt a *Control Panel/System/../Environment Variables* címszó alá kell beírni, célszerűen az *all user*-nek való változók közé, mert akkor mindenkinél élni fognak a beállítások.

1.2. Linux, Unix

1.10. Kérdés. Szeretném linux alatt futtatni az mosml-t, de nem tudom elindítani, mivel mindig hiányolja a *libm.so.5* fájlt. Mindent beállítottam, amit az *install.txt* írt. Mi lehet a probléma?

1.10. Válasz. ***Hallgató:*** Gondolom az, hogy vagy nincs *libm.so.5* a gépen, vagy nincs benne az *ld.cache*-ben, az *install.txt* pedig minden bizonnyal feltételezi, hogy a standard könyvtárak a helyükön vannak.

Próbálj rákeresni (pl. *locate*-tel), és megnézni, hogy benne van-e a */etc/ld.cache-ben.* Ha nincs, akkor javaslom az *LD_LIBRARY_PATH* beállítását a mosml futtatása előtt.

Még valami: ez nem a régi *libc* matematikai könyvtára? Ha egy régi mosml-t telepítesz egy új Linuxra, akkor fordulhat ilyen elő. Nézd meg a disztribúciódat, nincs-e benne valami *(libc)* kompatíbilitási csomag, vagy szerezz egy újabb mosml-t (lehet, hogy nincs ilyen binárisan), vagy fordíts egyet magadnak. Ez az utolsó mentsvár, de így biztosan működni fog.

2. fejezet

A Poly/ML-értelmező használata

2.1. Kérdés. A polyml-lel kapcsolatban van pár kérdésem. Az előadáson elhangzott, hogy a polyml-ben lehet debuggolni és valamilyen trace funkciót is tartalmaz. Ezek használatára vagyok kíváncsi. Meg arra, hogy a polyml-t hogyan lehet beépíteni az EMACS-ba?

2.1. Válasz. ***Oktató:*** A Poly/ML hibakeresési és nyomkövetési lehetőségeiről a *Hibakeresés, nyomkövetés* c. fejezetben olvashat.

A Poly/ML-t az emacs-ban ugyanazzal az *sml-mode* csomaggal használhatja, mint az mosml-t vagy az smlnj-t. A Poly/ML-t az emacs-ban úgy kell elindítania, hogy az *M-x run-sml* parancsnak *ML command*-ként (általában) a *poly* nevet írja be.

3. fejezet

Emacs és SML

3.1. Kérdés. Mi az az emacs editor? Azzal kéne használni valahogy az SML-t? Mert ha igen, honnan lehet megszerezni?

3.1. Válasz. ***Hallgató:*** Tudomásom szerint linux alá szövegszerkesztőnek indult, de kicsit már kinőtte ezt a mivoltát. Azért még szöveget is lehet vele szerkeszteni :)))

Egyébként nemcsak linux, de windoz alá is lehet gnuEmacs-ot vagy XEmacs-ot feltenni, lásd <http://www.emacs.org> és <http://www.xemacs.org>. Ha valaki tudja, hogyan kell rábírni, hogy „együttműködjon” az mosml-lel, akkor feltétlenül írja meg!! Talán ez a kisebbik rossz. . .

3.2. Kérdés. Hogyan kell az emacs-ban beállítani az sml-mode-ot?

3.2. Válasz. ***Hallgató:*** A dp-honlapról vagy az eredeti mosml-honlapról letölthető bináris csomagban benne van az *sml-mode*, a *<kicsomagolt tar>/mosml/utility/sml-mode-3.3b/README*-ben le van írva, hogyan telepíteni kell. Igazán nem bonyolult. Csak pár sort kell hozzáírni a *.emacs*-odhoz. Még egyszerűbb, ha debiant használsz: *apt-get install sml-mode*.

Ha többet akarsz, keresd a google-lal az *sml-mode* szöveget, válassz, töltsd le és telepítsd, ahogy leírják. Néhány tipp:

- A legfrissebb verzió: <http://www.iro.umontreal.ca/~monnier/elisp/>
- Korábbi: <ftp://ftp.research.bell-labs.com/dist/smlnj/contrib/emacs/sml-mode-3.9.5.tar.gz>
- Kicsit elavult (de magyar!): <http://dp.iit.bme.hu/mosml/doc/telepites-emacs-sml.txt>

Ha a telepítés kész, indítsd el *M-x run-sml [RET]*-tel az emacsban.

4. fejezet

Jótanácsok

4.1. Monológ. ***Hallgató:*** Ajánlanám mindenkinek a <http://www.dcs.napier.ac.uk/course-notes/sml/> sml-leírást. De most komolyan. Nagyon mayer… Példákon keresztül mutatja be a nyelvet, és abszolúte nem száraz, tele van poénokkal… :)

Az előadás jegyzete nem nagyon tartalmaz példákat, sajna így nem hiszem, hogy hamar meg lehetne tanulni egy nyelvet…

Nézzétek az oldalt, lőjjetek be egy mosml-t, és legyetek istenek! at least félistenek! :)

Oktató: Azoknak, akik angolul könnyen olvasnak, nagyon ajánlom a zéhára való felkészüléshez is, a címe: *A Gentle Introduction to ML*.

5. fejezet

Szintaxis, szemantika

5.1. Kérdés. Egy furcsa hibával küzdök. Ha valamelyik SML függvényben az `o` névvel hivatkozom egy elemre, akkor kiakad a program, és kapok egy `Ill-formed infix expression` hibaüzenetet.

5.1. Válasz. ***Hallgató:*** Az `o` név az SML-ben egy infix operátor neve.

5.2. Kérdés. Az alábbi függvénydefiníció az mosml szerint hibás. Miért?

```
- fun b((S, O, E):As, N) = (S, O, E)::b(As, N)
    | b([], N) = [];
! Toplevel input:
! fun b ((S, O, E):As, N) = (S, O, E)::b(As, N)
!                                      ^^
! Unbound value identifier: As
```

5.2. Válasz. ***Oktató:*** Én az mosml-től más hibaüzenetet kapok:

```
! Toplevel input:
!  fun b((S, O, E):As, N) = (S, O, E)::b(As, N)
!                  ^^
! Unbound type constructor: As
```

Én csak ezt a programrészletet adtam oda az mosml-nek, valószínű, hogy más környezetben kapta a fenti hibaüzenetet. Akárhogy is, a hiba az, hogy egyetlen `:`-ot írt a megjelölt helyen `::` helyett. Így az mosml típusmegkötésnek tekinti, márpedig az `As` típuskent nincs definiálva, és a definíció jobb oldala alapján nem is lehet típus.

5.3. Kérdés. Most egy függvényt nem értek és nem találok sehol, az `as`-t. Mit csinál? Hogyan kell használni?

5.3. Válasz. ***Oktató:*** Az `as` nem függvény, hanem kulcsszó, az ún. *réteges minta (layered pattern)* kulcsszava. Pl. az `(xxs as x::xs)` kifejezésben a teljes listára az `xxs` névvel, a lista fejére az `x`, a farkára az `xs` névvel lehet hivatkozni egy függvénydefinícióban annak a klóznak a törzsében, amelynek a fejében ez a kifejezés mintaként szerepel.

5.4. Kérdés. Miért van az, hogy a függvényparaméter (pl. az `exists`-nél) `fn`-nel megadva nem működik, csak ha külön „nevesítem”? Tehát az alábbi részlet hibás:

```
let fun felfele (Y,X,A) =
      X = O andalso Y > S andalso A > 0 andalso
        not (List.exists (fn (Y1,X1) =>
               X1 = X andalso Y1 < Y andalso Y1 <> S) Pontok)
```

míg a következő már jó:

```
let fun felfele (Y,X,A) =
      let F (Y1,X1) = (X1 = X andalso Y1 < Y andalso Y1 <> S)
      in X = O andalso Y > S andalso A > 0
           andalso not List.exists F Pontok
      end
```

5.4. Válasz. ***Hallgató:*** Nekem az első jónak tűnik, feltéve, hogy `S`-nek és `Pontok`-nak van értéke, ezzel szemben a második változat biztosan hibás, mert a `let` után kell még egy `fun` kulcsszó is. Emellett a `List.exists F Pontok` kifejezést be kell zárójelezni, különben precedenciahiba is van. Szóval a hiba nem itt van.

A kérdező folytatja: Pedig típushibával kidobta! Valami olyasmit irt, hogy `bool` nem egyezhet `(valami * valami ..) -> bool`-lal. Sajnos nem tudom pontosan idézni, és már átírtam. Az érdekes számomra az, hogy ha nem a könyvtári `List.exists`-et használom ugyanígy, hanem a saját függvényeimet, akkor síman megy. Hogy működik a típusellenőrzés, miért „szigorúbb” a beépítettekkel, mint a megírtakkal? Létezik ez? Nem értem, bár lehet, hogy csak elírtam valamit, és megint nem vettem észre…

„Ezzel szemben a második változat biztosan hibás”: igaz, de csak az emilben hagytam ki őket, a programban bent voltak, és úgy tényleg működött. Bár *matching warning*-ot kaptam arra is.

Hallgató: Valamit mégiscsak elfelejtettél lezárni. Azt javaslom, próbáld meg a fájlt darabokban odaadni az értelmezőnek. Legcélszerűbb felezéssel (először az első felét, ha nincs hiba, a háromnegyedét, ha van, a negyedét stb.)

A kérdező folytatja: Köszönöm, a módszer bevált, egy `let`-részben volt véletlenül a következő függvény előtt még egy, felesleges `let`…

Hallgató: „igaz, de csak az emilben hagytam ki őket…”: Csak azt tudom mondani, hogy biztosan nem a lambda-jelölés <-> nevesített függvény okozta a különbséget, valami más is lehetett ott még. Sajnos elég nehéz a hibát úgy megállapítani, hogy nem tudod a pontos kódot megmutatni…

A kérdező folytatja: Mindegy, a fő, hogy most működik.

6. fejezet

Kisbetű, nagybetű

6.1. Kérdés. A házi feladat beküldését nyugtázó levélben ezt a hibaüzenetet kaptam:

```
...
! local open Array2 Tcikcakk
!       ^^^^^^^^^^^^^^^^^^^
! Cannot find file Tcikcakk.ui
...
```

Pedig a programom itthon hibátlanul lefordult! Mi a baj vele?

6.1. Válasz. ***Oktató:*** Otthon nyilván Windows alatt dolgozik, a fogadórendszer pedig linuxos. A baj az, hogy `Tcikcakk` helyett `TCikcakk`-ot kell írnia, azaz a Cikcakk szót is nagy C-vel kell kezdenie, mert a struktúra nevét így írtuk. A kis- és nagybetűket, amint erre a sokszor felhívtuk már a figyelmüket az előadásokon, a unix/linux – a Windows-zal ellentétben – megkülönbözteti.

6.2. Kérdés. Sajnos nem tudom beadni a nagy házi feladatot már két napja, mert a tesztelőprogram mindig valamilyen más hibaüzenetet küld. Használom a `word` struktúra beépített függvényeit, és ha loadolni akarom, akkor azt mondja hogy *syntax error*, ha meg kihagyom a `load`-ot, akkor nem találja a szerveren a `word.ui`-t, pedig itt nálam lefut az összes tesztteset, a kiadott 29*22-es is 3,9 mp belül, és nem tudom, hogy mit kellene tennem, hogy ott is lefordúljon. Legutóbb ezt kaptam:

```
! Cannot find file word.ui
```

máskor meg ezt:

```
! load "Word";
!
! ^^^^
! Syntax error.
```

attól függően, hogy hogyan olvastatom be a `word`-öt.

6.2. Válasz. ***Oktató:*** A `load`-ot nem szabad használnia, mert az csak interaktív módban használható, a fogadóprogram viszont lefordítja a programot.

A Windows-zal ellentétben a Unix/linux rendszerek megkülönböztetik a fájlnevekben a kis- és nagybetűket, tehát nem mindegy, hogy `word`-öt vagy `Word`-öt ír. Ha az alapkönyvtárakra pontosan olyan néven hivatkozik, ahogyan az az *MOSML Reference Manual*-ben le van írva (`Word`, `TextIO`, `Char` stb.), akkor Unix/linux alatt is működőképes lesz a programja. Más fájlnevek esetén is ügyelnie kell a kis- és nagybetűk használatára. Erre az apró különbségre sokszor felhívtuk már a figyelmet.

Az mosml mind Windows, mind unix/linux alatt megkülönbözteti a kis- és nagybetűket, ha tehát pl. a `word` szót típusnévként nem csupa kisbetűvel írja, az mosml értelmező és az mosmlc fordító is hibát jeleznek.

7. fejezet

Deklaráció

7.1. Kérdés. A következő probléma merült fel a nagyházimmal. Az elkészült program mosml-lel működik:

```
> val felhok = fn :
  int list * int list * refpont list ->
                              (int * int * int * int) list list
- felhok([7, 7, 0, 6, 6, 8, 4],
              [~1, 5, 2, 3, 3, 0, 2, ~1, 4, 2, 2, 4, 2, ~1], []);
> val it =
    [[(1, 1, 2, 3), (1, 8, 2, 9), (1, 11, 2, 12), (4, 1, 6, 2),
      (4, 4, 6, 5), (4, 12, 5, 13), (6, 7, 7, 10)],
     [(1, 2, 2, 3), (1, 7, 2, 8), (1, 10, 2, 12), (4, 1, 6, 2),
      (4, 4, 6, 5), (4, 8, 7, 9), (6, 12, 7, 13)]]
    : (int * int * int * int) list list
```

Ha viszont a keretprogramot akarom használni, a következő hibaüzenetet kapom:

```
C:\test>mosmlc -c TFelhok.sml Felhok.sig Felhok.sml KFelhok.sig
                      KFelhok.sml
! Interface mismatch: the implementation of unit Felhok
! does not match its interface, because ...
! Scheme mismatch: value identifier Felhok.felhok
! in the unit Felhok is specified with type scheme
!   val felhok :
  int list * int list * refpont list ->
                      (int * int * int * int) list list
! in the interface
! but its declaration has the unrelated type scheme
!   val felhok :
  int list * int list * refpont/1 list ->
                      (int * int * int * int) list list
! in the implementation
```

```
! The declared type scheme should be at least as general
! as the specified type scheme
```

Mi lehet a hiba oka?

7.1. Válasz. ***Oktató:*** A bajt az okozza, hogy a

```
datatype refpont  = f of sor * oszlop | e of sor * oszlop
```

deklaráció (és a hozzátartozó `type`-deklarációk) nemcsak a `TFelhok.sml` állományban, hanem a `Felhok.sml` állományban is előfordulnak. Ha pl. a

```
type sorosszeg    = int
type oszloposszeg = int
type sor          = int
type oszlop       = int
datatype refpont  = f of sor * oszlop | e of sor * oszlop
```

deklarációsorozatot kiértékelteti az mosml értelmezővel, majd a

```
datatype refpont  = f of sor * oszlop | e of sor * oszlop
```

deklarációt újra kiértékelteti, akkor az mosml ezt válaszolja:

```
> New type names: =refpont/1
  datatype refpont =
  (refpont/1,con e : int * int -> refpont/1,
             con f : int * int -> refpont/1)
  con e = fn : int * int -> refpont/1
  con f = fn : int * int -> refpont/1
```

vagyis a `refpont` egy új típus neve lett (erre utal a `refpont/1` jelölés), semmi köze a korábbi azonos nevű típushoz.

Ez még önmagában nem lenne baj (csak felesleges), ha a `felhok` függvény specifikációjában a `TFelhok` struktúrában deklarált típusokra hivatkozna a teljes nevükkel, pl. így:

```
fun felhok (x : TFelhok.radarkep) : TFelhok.felhok list = ...
```

De hibát okoz, ha a `TFelhok` szelektort elhagyja:

```
fun felhok (x : radarkep) ...
```

8. fejezet

Kifejezés kiértékelése

8.1. Kérdés. Adott a következő értékdeklaráció:

```
val x = List.filter (fn true => false | _ =>true)
                          [3.0>4.0, 4.7>2.4, ~3.0<=0.4]
```

erre az mosml azt írja ki, hogy `x = [false]`. De miért? Mit csinál a `List.filter`-ben megadott feltétel?

8.1. Válasz. ***Oktató:*** A teljes kifejezés a listában lévő igaz értékű elemeket eldobja, a hamis értékűeket kigyűjti.

A predikátum (`List.filter` első argumentuma) éppen ezt fogalmazza meg: ha az elem `true`, eldobja, ha `false` (pontosabban bármi más, vö. `_ => true`, de most csak `true` lehet), akkor kigyűjti.

A pontosság kedvéért: a `List.filter` első argumentumaként nem egy *feltételt*, azaz egy `bool` típusú *kifejezést*, hanem egy `predikátumot`, azaz egy `bool` típusú eredményt adó *függvényt* vár.

8.2. Kérdés. Az SML kisházi megoldása közben a honlapon megadott példák közül mind lefutott, az utolsót kivéve, amely a következő hibaüzenetet adta:

```
- dadogo 123456789;
! Uncaught exception
! Overflow
```

Feltételezem, hogy az mosml ennél nagyobb `int` típusú számokat is képes kezelni, ezért nem értem a dolgot.

8.2. Válasz. ***Oktató:*** 123456789 még valóban belefér az `int` tartományba. De ha megszorozza pl. 9-cel, ezt kapja:

```
- 123456789 * 9;
! Uncaught exception:
! Overflow
```

8-cal még éppen megszorozhatja. Feltehetően olyan műveletet végez a számmal, amellyel kilép az `int` tartományból.

8.3. Kérdés. A következő függvények betöltésénél a *Warning: calling polyEqual* hibaüzenetet kapom az smlnj-től. Az iránt érdeklődnék, hogy ez pontosan mit jelent.

```
open List;
fun dadoge [] = true
  | dadoge a = if hd a = last(take(a, length a div 2 + 1))
                    andalso parose a = true
               then dadoge(drop(take(a, length a div 2), 1) @
                           drop(drop(a, length a div 2), 1))
               else false
and parose a = length a mod 2 = 0;
```

8.3. Válasz. ***Oktató:*** Lássuk akkor pontosabban a figyelmeztetést (csak *figyelmeztet*, nem *hibát jelez*!) smlnj-vel:

```
... Warning: calling polyEqual
val dadoge = fn : ''a list -> bool
val parose = fn : ''a list -> bool
```

majd mosml-lel:

```
> val ''a dadoge = fn : ''a list -> bool
  val ''a parose = fn : ''a list -> bool
```

Az smlnj tehát akkor is *figyelmeztet*, ha valaki az egyenlőségvizsgálatot *politípusú* értékeken végezteti el. A következő nagyon egyszerű esetben ugyanez a helyzet:

```
- fun eq (x,y) = x=y;
... Warning: calling polyEqual
val eq = fn : ''a * ''a -> bool
```

Ha típusmegszorítást használunk, mindent rendben lévőnek talál:

```
- fun eq (x,y:int) = x=y;
val eq = fn : int * int -> bool
```

Ezzel szemben egyáltalán nem engedi meg a következőt:

```
- fun eq (x,y:real) = x=y;
... Error: operator and operand don't agree
           [equality type required]
  operator domain: ''Z * ''Z
  operand:         ''Z * real
  in expression:
    x = y
```

ui. `real` típusú értékek egyenlősége az `=` operátorral az smlnj használatakor *nem* vizsgálható.

Helyette a `Real.==` vagy a `Real.?=` függvény használható. Pl.

```
fun eq (x,y) = Real.==(x,y);
val eq = fn : real * real -> bool
```

A `Real.==` és a `Real.?=` műveletek kielégítik az *IEEE standard 754-1985* szerint előírt specifikációt (lásd: <http://www.standardml.org/Basis/real.html#REAL:SIG:SPEC>).

Most már az is érthető, hogy politípusú értékek egyenlőségvizsgálatakor miért figyelmeztet az smlnj: előfordulhat, hogy a létrehozott függvény a későbbiekben olyan értékek egyenlőségét akarja megvizsgálni, amelyekre az egyenlőségvizsgálat nem végezhető el.

Az mosml-ben a `Real.==` ekvivalens az `=` operátorral, ha az utóbbi operandusai `real` típusúak, a `Real.?=` eredménye pedig mindig `false`, mert az IEEE standard szerinti számábrázolások az mosml-ből hiányoznak.

8.4. Kérdés. Ha egy függvényben használok egy belső függvényt (`let fun...`), amelyben az egyenlőségjel jobb oldalán (azaz a belső függvény törzsében) feltételvizsgálatra van szükségem, akkor ennek `else` ágában hogyan léphetek ki a belső függvényből?

8.4. Válasz. ***Oktató:*** Ugyanúgy, mint a `then` ágából: a megfelelő típusú értékkel.

Hangsúlyozom, hogy az SML-ben az `if... then... else`: *kifejezés*, mégpedig egy feltételtől függően így vagy úgy kiértékelendő kifejezés. Ha az `else` ágban nincs rekurzív hívás, az esetleges műveletek elvégzése után a függvény kiértékelése is befejeződik, azaz az SML visszatér a függvényhívásból.

A kérdező folytatja: Ha van több lehetőség is, melyik mit jelent pontosan ?

Oktató: Ezt nem értem: ha az `if... then... else... if...else ...` szerkezetre gondol, akkor a kiértékelés mindig az `if` után álló predikátumtól függ. Ha a függvény klózaira gondol, akkor pedig a függvényfejben megadott mintáktól.

Figyelem: mind a `then` és `else`-ágakban, mind a függvény klózaiban álló kifejezések függetlenek egymástól!

A kérdező folytatja: Valójában a belső függvényből való kilépéssel együtt a külső függvényt is be szeretném fejezni, és a belsőből való kilépésig kiszámított eredményeket visszadni. Remélem, nagyjából követhető volt a probléma leírása, és van ötletetek a megoldására.

Oktató: Ennek semmi akadálya: ha a belső függvény visszatérési értékével egyúttal az őt meghívó függvényből is kilép, akkor a külső függvény által visszaadott érték meg fog egyezni a belső függvény által visszaadott értékkel (a belső függvény *visszatérési értékével).*

Abból, amit leírt, azért nem nagyon értem, hogy mi a problémája. Ha esetleg arra gondol, hogy valamilyen `return v` vagy `exit if` parancs van-e az SML-ben, akkor erre *nem* a válasz, mert nincs rájuk szükség. Ha megnézi az előadásokon bemutatott példákat, sok mintát talál.

8.5. Kérdés. Adott egy programrészlet:

```
fun megy m n =
  let
    fun megy2 sor1 oszlop1 =
     if sor1 = n andalso oszlop1 > m
     then   (* legalján vagyunk, visszatérünk valamivel *)
          [[(1,1,1,1)]]
     else if sor1 <= n andalso oszlop1 > m
     then   (* sor végén vagyunk *)
          megy2 sor1+1 1
     else   (* sorban vagyunk *)
          megy2 sor1 oszlop1+1
  in
      megy2 1 1
  end;
```

A lényege az, hogy végiglépkedek egy mátrixon bal fentről jobbra le. Fordításánal ezt a hibát kapom:

```
! Toplevel input:
!           megy2 sor1+1 1
!           ^^^^^^^^^^
! Type clash: expression of type
!   'a -> (int * int * int * int) list list
! cannot have type
!   (int * int * int * int) list list
```

De miért? Miért akarja az

```
'a -> (int * int * int * int) list list
```

típust adni neki?

8.5. Válasz. ***Oktató:*** A precedenciákkal, pontosabban a függvényalkalmazás *móhóságával* van baj: a függvényeknek argumentumként átadott *összetett* kifejezéseket zárójelbe kell tenni, azaz:

```
...
     then   (* sor végén vagyunk *)
          megy2 (sor1+1) 1
     else   (* sorban vagyunk *)
          megy2 sor1 (oszlop1+1)
...
```

8.6. Monológ. ***Hallgató:*** Megoldottam a listára küldött példát mind mohó, mind a lusta módszerrel. Az eredmény egyezik, csak nem vagyok biztos benne, hogy minden lépés, ill. azok sorrendje helyes-e. A hibákat, észrevételeket légyszi küldjétek vissza! A függvény:

```
fun  f g [] = []
  | f g (x::xs) = map g x :: f g xs;
```

Egy alkalmazása *lusta* kiértékeléssel:

```
f chr [[2*5, 7 div 3], [round 8.0], []] =
   chr [2*5, 7 div 3] :: f chr [[round 8.0], []] =
   chr [2*5, 7 div 3] :: chr [round 8.0] :: f chr [] =
   chr [2*5, 7 div 3] :: chr [round 8.0] :: [] =
   [chr 2*5, chr 7 div 3] :: chr [round 8.0] :: [] =
   [chr 2*5, chr 7 div 3] :: [chr round 8.0] :: [] =
   [chr 10, chr 7 div 3] :: [chr round 8.0] :: [] =
   [chr 10, chr 2] :: [chr round 8.0] :: [] =
   [chr 10, chr 2] :: [chr round 8.0] :: [] =
   [#"\n", chr 2] :: [chr round 8.0] :: [] =
   [#"\n", #"\^B"] :: [chr round 8.0] :: [] =
   [#"\n", #"\^B"] :: [chr 8] :: [] =
   [#"\n", #"\^B"] :: [#"\b"] :: [] =
   [#"\n", #"\^B"] :: [#"\b"] =
   [[#"\n", #"\^B"], [#"\b"]]
```

Ugyanez *mohó* kiértékeléssel:

```
f chr [[2*5, 7 div 3], [round 8.0], []] =
   f chr [[10, 7 div 3], [round 8.0], []] =
   f chr [[10, 2], [round 8.0], []] =
   f chr [[10, 2], [8], []] =
   f chr [[10, 2], [8], []] =
   [chr 10,2] :: f chr [[8], []] =
   [#"\n", 2] :: f chr [[8], []] =
   [#"\n",chr 2] :: f chr [[8], []] =
   [#"\n",#"^B"] :: f chr [[8], []] =
   [#"\n",#"\^B"] :: chr [8], f chr [] =
   [#"\n",#"\^B"] :: [#"\b"] :: f chr [] =
   [#"\n",#"\^B"] :: [#"\b"] :: [] =
   [#"\n",#"\^B"] :: [#"\b"] :: [] =
   [#"\n",#"\^B"] :: [#"\b"] =
   [[#"\n",#"\^B"],[#"\b"]]
```

9. fejezet

Függvény, rekurzió

9.1. Kérdés. Lenne egy egyszerű kérdésem. Egy függvény végeredményét hogy lehet negálni?

9.1. Válasz. ***Oktató:*** SML-ben a `not` függvény alkalmazásával.

9.2. Kérdés. Van valakinek 5lete, hogy lehetne egy eljárás több klózában ugyanazokat a lokális változókat használni?

Háromféleképp próbáltam, de nem ment (egyszer az `end`-re, aztán a `|`-ra (vonásra) mondott syntax error-t). Fiktív példákkal:

```
fun Q (A:As,[]) = bla
  | Q (A:As,B:Bs) =
          let A = 1+1
          in
             if A = blablabla ...
          end
  | Q ([],B:Bs) =
          let A = 1+1
          in
             if A = blablabla ...
          end
```

Az 1. példában a legelső `end`-nel volt gond.

```
fun Q (A:As,[]) = bla
  | Q (A:As,B:Bs) =
          let A = 1+1
          in
             if A = blablabla ...
  | Q ([],B:Bs) =
             if A = blablabla ...
          end
```

A 2. példában a második `|`-sal volt gond.

```
let A = 1+1
  in
     fun Q(A:As,[]) = bla
       | Q(A:As,B:Bs) =
            if A = blablabla ...
       | Q([],B:Bs) =
            if A = blablabla ...
end
```

A 3. példában a `fun`-nal volt gond.

9.2. Válasz. ***Oktató:*** *Látszólag* tényleg ezekkel volt a gond, a szintaxisellenőrző legalábbis ezeknél vette észre, hogy valami hézag van. A nagyon elemi hibák azonban máshol vannak.

Hibák az 1. példában:

- A `let` kulcsszó után deklarációnak kell jönnie, az adott esetben a `val` kulcsszóval kezdődő értékdeklarációnak: `val A = ...`
- A `:` (kettőspont) a *típusmegkötés* jele, a `::` (négyespont) a lista konstruktoroperátora. A példában a négyespontot kellett volna használnia.
- A `| Q([],B:Bs) = ...` klózban a bal oldalon nem vezeti be az `A` paramétert, ezért a jobb oldali kifejezésben sem használhatja! A klózfejek *egymástól függetlenek*, amit az egyik klózban paraméterként használ, azt a másik klózból nem látható, nem használható. Az adott minta az üres listára illeszkedik, és csak arra, semmi köze ahhoz, hogy az előző klóz korábban milyen mintára illeszkedett.

  Ebből az is következik, hogy nem tud olyan lokális nevet bevezetni az egyik klózban, amelyik egy másik klózban is használható lenne.

A 2. példában is felsoroltak mellett még egy hiba van:

- A `let D in E end` egy kifejezés *(kifejezés lokális deklarációval)*, az `if B then F else G` egy másik kifejezés *(feltételes kifejezés)*. `E`, `F` és `G` helyébe tetszőleges SML-kifejezés írható, többek között tetszőleges kifejezés lokális deklarációval vagy tetszőleges feltételes kifejezés, de a kifejezések kettő nem lapolhatják át egymást, mint ebben a példában.

A 3. példában a korábbiakhoz képest nincs új hiba, ám a külső `let`-kifejezésben hiába deklarálja (deklarálná, ha nem lenne hibás a deklaráció!) a lokális `A` nevet, azt a `Q` függvény első és második klózában bevezett `A` paraméterek, amelyek maguk is egymástól független dolgokat jelölnek, elfedik (vö. láthatóság és érvényességi kör). Egyedül a harmadik klózban látszik a külső `let`-kifejezésben bevezett `A` név, de tartok tőle, hogy nem ezt akarta elérni.

E hosszú történet talán legfontosabb tanulsága, hogy bárkinek, mielőtt programírásba fogna, érdemes megismerkednie az SML-kifejezések elemi szintaxisával. Ajánlom mindenkinek a figyelmébe az SML szintaxisát ismertető fejezetet a jegyzetben.

9.3. Kérdés. Az lenne a kérdésem, hogy ha SML-ben két függvény egymást hívja, akkor mit kell csinálni, hogy a fordító elfogadja. Ugye az egyik mindenképp olyan függvényt hív meg, amelyet még nem definiáltunk. Meg lehet valahogy mondani a fordítónak, hogy ilyen nevű függvényt definiálni fogunk később?

9.3. Válasz. ***Oktató:*** Kölcsönösen rekurzív függvényeket és más egymástól függő értékeket az ún. *egyidejű deklarációval* hozhatunk létre. Erre való az `and` kulcsszó. Pl.

```
fun odd  0 = false
  | odd  n = even(n-1)
and even 0 = true
  | even n = odd(n-1)
```

Az `and` más deklarációsorozatokban is használható, nemcsak kölcsönösen rekurzív vagy egyéb, egymástól függő esetekben. Pl.

```
val harom = 3
and negy  = 4
and ot    = 5
fun f1 x  = x * 2
and f2 x  = x div 2
```

abban különbözik a

```
val harom = 3
val negy  = 4
val ot    = 5
fun f1 x = x * 2
fun f2 x = x div 2
```

deklarációsorozattól, hogy az első sorozatban az első hármat, majd pedig a második kettőt *egyszerre* értékeli ki az mosml, míg a második sorozatban minden sort külön-külön, azaz szekvenciálisan értékel ki. Ha kiírjuk a pontosvesszőket (amit egyébként deklarációk között nem kötelező kitenni), akkor jobban látszik a különbség, az `and` elé ugyanis *nem szabad* pontosvesszőt tenni, ha kitesszük, hibajelzést kapunk:

```
val harom = 3
and negy  = 4
and ot    = 5;
fun f1 x  = x * 2
and f2 x  = x div 2;
```

és

```
val harom = 3;
val negy  = 4;
val ot    = 5;
fun f1 x = x * 2;
fun f2 x = x div 2;
```

Az egyidejű deklaráció felhasználható kötések felcserélésére is, pl. a

```
val alma = "szeretem"
val korte = "nemszeretem"
```

vagy a

```
val alma = "szeretem" and korte = "nemszeretem"
```

deklaráció után a

```
val alma = korte and korte = alma
```

deklaráció megcseréli a két kötést, míg a

```
val alma = korte
val korte = alma
```

deklaráció a szekvenciális kiértékelés miatt nem a remélt eredményhez vezet.

9.4. Kérdés. Hogyan oldható meg úgy egy függvénydeklaráció, hogy ne kelljen különválasztanom a `nil` esetet és a listaátadás esetét? Azaz:

```
fun valami (nil) = ...
  | valami (l)   = ...
```

így jó, de nekem az kellene, hogy

```
fun valami (l) = ...
```

esetén is működjön, és `l` értéke legyen `nil`, ha arról van szó. Ha ilyent írok be, akkor mindig kiírja, hogy nem fedem le az összes esetet, pedig én szeretném. :)

9.4. Válasz. ***Oktató:*** Ez valahogy nem stimmel. Az `l` mint minta *mindenre* illeszkedik, tehát nem okozhat figyelmeztetést. Annak valami más lehet az oka. Megjegyzés: a zárójel a `nil` és az `l` körül felesleges (persze nem hiba).

Kérés: ha "hogyan kell megoldani, hogy..." vagy "miért az a válasz, hogy..." jellegű kérdésük van, kérjük, mellékeljék a megfelelő (rövid, de a kérdéses viselkedést produkáló) programrészletet és az értelmező válaszát, ui. csak akkor tudjuk pontosan megmondani, hogy mi lehet a gond. Ezek nélkül legfeljebb találgathatunk, vagy széttárhatjuk a kezünket, mint most.

10. fejezet

Minta, mintaillesztés

10.1. Kérdés. Az alábbi `szomszed1 (c,d)` függvényt az mosml a következő figyelmeztetéssel fordítja le: `Some cases are unused in this match`.

A függvény feladata az, hogy egy listába gyűjtse azokat a mátrixelemeket, amelyekre egy adott `(a,b)` koordinátájú mátrixelemről ráléphetek (`a` a sor-, `b` az oszlopindex). A kódot az órán bemutatott `joszam10 (a,b)` függvényhez hasonlóan próbáltam megírni.

Vázlatosan (a kipontozott helyeken a megértéshez nem szükséges értékdeklarációk és feltételvizsgálatok vannak):

```
fun szomszed (a, b) =
      let val alsosor = a+1
          val szelsosor = b+1
          val szsorutan = b+2
          val kozepa = a
          val kozepb = b
          fun szomszed1 (alsosor, szsorutan) = []
            | szomszed1 (c, szsorutan) =
                                 szomszed1(c+1, szelsosor-2)
            | szomszed1 (kozepa, kozepb) =
                                szomszed1(kozepa, szelsosor)
            | szomszed1 (c, d) =
                 ................
                 ................
                if ........ then  ujelem :: szomszed1(c, d+1)
                                  else  szomszed1(c, d+1)
    in
       szomszed1 (a-1,b-1)
    end
```

10.1. Válasz. ***Oktató:*** A függvénydefinícióban formális paraméterként használt kifejezések olyan *kifejezésminták*, röviden *minták*, amelyekre az mosml majd megpróbálja illeszteni a függvény alkalmazásakor használt aktuális paramétereket (argumentumokat); ezt nevezzük *mintaillesztésnek*. Az SML-kifejezéseknek csak egy részhalmaza használható mintaként. To-

vábbi részletek az SML-jegyzet *Az SML alapnyelv szintaxisa* c. függelékében, valamint a *Moscow ML Language Overview*-ban találhatók. Összefoglalva:

Minta	*Magyarázat*
Mindenesjel (_)	Mindenre illeszkedik; a definíció jobb oldalán nem használható
Állandó	Csak a meghatározott értékre illeszkedik
Név (azoosító)	Mindenre illeszkedik
Adatkonstruktor	Csak meghatározott szerkezetű, ill. tartalmú értékre illeszkedik
Rekord	Csak adott szerkezetű rekordra illeszkedik
Ennes (nullas is)	Csak adott szerkezetű ennesre illeszkedik
Lista	Csak adott szerkezetű listára illeszkedik

A konkrét kérdésre válaszolva: a `szomszed1` függvénynek már az első klózában olyan minta van, amely az összes lehetséges esetet lefedi, ezért a többi klóz kiválaszására soha nem kerül sor. (Egyébként a `val alsosor = a+1` értékdeklarációnak nincs semmi hatása a mintára.)

A kérdező folytatja: Akkor hogyan lehet ezt a problémát megoldani?

Oktató: Mint láttuk, csak bizonyos kifejezések lehetnek minták az SML-ben, aritmetikai kifejezések és szinonimák nem. Ezért a `szomszed1` függvényben – úgy, ahogyan a programrészletből vélhetően használni szeretné – sem `alsosor`, sem `kozepa` nem lehet minta, de nem lehet minta a fentiek szerint `a+1` sem.

Egy aritmetikai kifejezés értéke csak a függvény törzsébe beírt kóddal vizsgálható meg, pl. az adott esetben az `x` aktualis értéke a `szomszed1` függvény törzsében így: `if x = alsosor then ...`

A mintaillesztésnek erős korlátai vannak az SML-ben. Más funkcionális nyelvekben (pl. gofer, clean, haskell, Alice) a mintaillesztés sokkal rugalmasabb.

10.2. Kérdés. Meg tudná mondani valaki, hogy a következő függvényre miért kapom a `Pattern matching is not exhaustive` hibaüzenetet?

```
fun uj_satrak ((fx,fy), ujak, iranyeg, [],[]) =
                                              (ujak, iranyeg)
  | uj_satrak ((fx,fy), ujak, iranyeg,
                       (ix,iy)::iranyok, egt::egtajak) = ...
```

10.2. Válasz. ***Oktató:*** Azért, mert a függvény azokra az esetekre nem illeszkedik (nincs rájuk minta), amikor az `iranyok` és az `egtajak` listák közül az egyik üres, a másik meg nem.

Amúgy ez nem hibaüzenet (error message), csak figyelmeztetés (warning), de erősen ajánljuk, hogy a figyelmeztetések okát is szüntesse meg a programjaiban, mert eltakarhatják az igazi hibákat.

Talán arra gondolt, hogy nem fordulhat elő olyan eset, amikor az `iranyok` és az `egtajak` közül az egyik üres, a másik meg nem. Ha ebben 100%-ig biztos, akkor a két sor megfordításával és apró módosításával (`[]` helyett a _ minta használatával mindkét esetben) a figyelmeztetés oka megszüntethető:

```
fun uj_satrak( (fx,fy), ujak, iranyeg,
                       (ix,iy)::iranyok, egt::egtajak) = ...
  | uj_satrak( (fx,fy), ujak, iranyeg, _, _) =
                                          (ujak, iranyeg);
```

De biztonságosabb, ha minden esetet külön kezel, és kivételt jelez hiba esetén. A kivétel jelzését főleg a program élesztése közben tanácsoljuk, később kiszedhető a programból.

10.3. Kérdés. Meg tudja mondani valaki, hogy miért kapok `Pattern matching is not exhaustive` *warning*-ot:

```
fun megfelel (Ss, Os, Fasor, MX, MY,
              (hanyadik, (sx, sy)::satrakK, _, AS,AO)) =
      if sx >= 1 andalso sx <= MX andalso
           sy >= 1 andalso sy <= MY andalso
           nemszomszed((sx,sy),satrakK) andalso
           mennyiseg_ok(AS,Ss, hanyadik) andalso
           mennyiseg_ok(AO, Os, hanyadik) andalso
           not(Tage((sx,sy),Fasor))
      then true
      else false
  | megfelel (Ss, Os, Fasor, MX, MY, (0,[],_,AS,AO )) =
                                                    false
  | megfelel ([], _, _, _, _, _) = false
  | megfelel (_, [], _, _, _, _) = false
  | megfelel (_, _, [], _, _, _) = false;
```

10.3. Válasz. ***Oktató:*** Mivel a kódrészlet nem teljes, csak tippelni tudok. Valószínűleg arra figyelmeztet, hogy nincs lefedve a következő eset: az `Ss`, `Os` és `Fasor` listák egyike sem üres, miközben az első klózban `(sx, sy)::satrakK` mintával jelölt lista mégiscsak üres. Egyébként már több ilyen kérdésre válaszoltunk, érdemes azokat is elolvasnia.

Néhány további tanács a programrészlethez:

- Az utolsó három sor egybevonható, ha a le nem fedett esetben is `false`-ot kell eredményül adnia a függvénynek; a minta egyetlen _lehet, hiszen az SML-ben minden függvény egyargumentumú.
- Az `if B then true else false` kifejezés felesleges bonyolítás, hiszen az eredménye a `B` értékével azonos! Úgy látszik, nem elégszer ismétlem, pedig minden félévben vagy egy tucatszor elmondom. Az ilyen kódrészlet a vizsgán pontlevonással jár!

10.4. Kérdés. Amikor betöltöm a nagyházimat az mosml-be, a következő üzenetet kapom:

```
! ....kitoltes(array,y,x,maxy,maxx,SOL,OOL,RefPL,1) =
!     if array<>[] then
```

```
!     if x<maxx andalso y<=maxy then
!         kitoltes(check1(addelement(array,y,1),
!                         ..........
!         kitoltes(check0(addelement(array,y,0),
                          ..........
!      else [addelement(array,y,0)]
!      else [].
! Warning: pattern matching is not exhaustive
```

A program utána fut, tehát nem hiba, hanem „csak” figyelmeztetés. Az a baj, hogy valszeg emiatt a hiba miatt bugos a nagyházim, és nem tudom, hogy mit jelent ez az üzenet. Tulajdonképpen elég lenne azt tudnom, hogy mikor ad ilyen figyelmeztetést az mosml, mert abból már ki tudom találni, hogy mi a hiba.

10.4. Válasz. ***Oktató:*** Ezt már több előadáson elmondtam, több dián le van írva, de röviden megismétlem.

Egy függvényben a paraméter(ek) összes lehetséges értékkombinációjára kell egy-egy klózt írni, különben a *Match* (lefedetlen esetek) figyelmeztetést írja ki az értelmező. Csak figyelmeztetés, mert előfordulhat, hogy az adott környezetben az adott kombináció sohasem fordulhat elő, de biztosabb ilyenkor is az összes lehetséges esetre klózt írni.

Gondolja át, hogy milyen eset(ek)et nem fedett le, és legalább valami értelmes hibaüzenetet írjon ki az eddig lefedetlen esetekben, amelyből következtethet a hiba okára. Ha jobb ötlete nincs, írjon egy minden esetet lefedő klózt (`... _ = ...`), és írja ki a képernyőre az adatok szignifikáns részét. Ez persze csak otthon, teszteles közben segíthet. Ha kell, bővítse a tesztesetek körét.

11. fejezet

Lokális érvényű deklarációk

11.1. Kérdés. Nem tudja vki, hogy pontosan mi a különbség a `let` és a `local` között??

11.1. Válasz. ***Oktató:*** A `let` olyan kifejezés, a `local` olyan deklaráció, amely felhasznál egy vagy több lokális érvényű deklarációt, röviden lokális deklarációt. Remélem, a kifejezés és a deklaráció közötti különbséget ismerik.

Egyszerű példák:

```
val x = let val a = 2 in a end;
local val a = 2 in val x = a end;
```

11.2. Kérdés. Nagyon megköszönném, ha valaki egy teljesen konkrét példán keresztül szemléltetve pontosan elmondaná, hogy mi a különbség a lokális kifejezés és lokális deklaráció között, tehát mi az, amit csak az egyikben szabad, a másikban nem.

11.2. Válasz. ***Oktató:*** Először is az elnevezéseket pontosítom. Korábban, többek között a jegyzet korábbi kiadásaiban valóban lokális kifejezésnek és lokális deklarációnak neveztük a `let`-kifejezést és a `local`-deklarációt, de ezek pontatlanok. Pontosabban fejezik ki a lényeget a *kifejezés lokális érvényű deklarációval* és a *deklaráció lokális érvényű deklarációval* elnevezések. *Lokális érvényű* helyett röviden *lokálist* is mondhatunk. És akkor most lássuk a jelentésüket és használatukat!

Közös mindkettőben, hogy a lokálisan, azaz a `let`, ill. a `local` és az `in` kulcsszók között) deklarált neveket a külvilág nem látja. A legfontosabb különbségek a következők.

Amikor egy függvénydefinícióban alkalmazzuk a `let`-kifejezést, a `let` és az `in` szavak közötti deklarációk a függvénydefiníció fejrésze *után* állnak a szövegben, ezért *látják* a függvényfejben szereplő paramétereket. A `let`-kifejezés `in` és `end` közötti részében *pontosan egy* kifejezésnek kell lennie.

Amikor egy függvénydefinícióban alkalmazzuk `local`-deklarációt, a `local` és az `in` szavak között álló deklarációk a függvénydefiníció fejrésze *előtt* állnak a szövegben, ezért *nem látják* a függvényfejben szereplő paramétereket. A `local`-deklaráció `in` és `end` közötti részében *legalább egy* deklarációnak kell lennie.

Példák:

```
local fun g x = 3 * x
in
   fun f (y, z) = (g y, g z)
end
```

Megjegyzés: A `g` függvény nem látja `y`-t, sem `z`-t, ezért a törzsében nem hivatkozhat rájuk.

```
fun f (y, z) =
        let fun g x = 3 * x * z
        in
           g y
        end
```

Megjegyzés: A `g` függvény deklarációja látja `y`-t is, `z`-t is, ezért a törzsében hivatkozhat rájuk.

12. fejezet

Nem frissíthető és frissíthető változók

12.1. Kérdés. Mi a különbség pontosan az `=` és `:=` között? Szeretném azt is megtudni, hogy a háttérben pontosan mit csinál az interpreter, vagyis pl. mi történik `val v1 = v2`-vel? Csak egy mutatót rak `v2` értékere? Mi lesz, ha ezután `val v1 = 4`-et adunk? Valahol leteszi a 4-est, és átirányítja a `v1` mutatóját?

12.1. Válasz. ***Oktató:*** Az *egyenlőségjel* (`=`) egy deklarációban a deklarálandó nevet választja el a névhez kötendő kifejezéstől. A *legyen-egyenlő jel* (`:=`) egy értékadásban választja el a változó nevét attól az értéktől, amelyet a változó által megnevezett memóriacellába kell beírni.

Az SML-ben a `val` deklarációval létrehozott dolgot nem változónak, inkább *névnek* (name) vagy *azonosítónak* (identifier) nevezzük, hogy megkülönböztessük az imperatív nyelvek változófogalmától. Az SML-beli név a matematikai változó fogalmának felel meg, ezért szokták változónak (variable) is nevezni, de ez, mint írtam, félrevezető. Ha mindenképpen változónak akarjuk nevezni, nevezzük *funkcionális változónak*. A funkcionális változó *nem frissíthető* (szemben az imperatív nyelvek frissíthető változójával), és nem értékadással, hanem a deklaráció által létrehozott *kötéssel* kap értéket.

A név tehát az SML-ben a deklaráció jobb oldalán álló érték megnevezésére, azonosítására szolgál, azaz az érték egy *szinonimája.* Az mosml értelmező a nevet a hozzá kötött értékkel együtt egy táblában tárolja; ha a deklarációban egy kifejezést kötünk egy névhez, akkor az mosml a mohó kiértékelésnek megfelelően azonnal kiértékeli a kifejezést, és az eredményül kapott értéket írja be a táblába. Egy név szinonima-volta azt jelenti, hogy valahányszor a név előfordul egy kifejezésben, az értelmező a nevet azonnal lecseréli a hozzá kötött értékre. Mutatóról, hivatkozásról tehát ilyen esetben szó nincs.

Lássuk akkor a példákat arra, hogy mi történik a háttérben! A `val v1 = v2` deklaráció hatására az mosml bejegyzi a táblába a `v1` nevet és hozzá köti a `v2` értékét (nem egy mutatót, nem a `v2` nevet, hanem a `v2` névhez kötött értéket). Nyilvánvaló, hogy ha a `v2`-nek nincs értéke, a deklaráció nem értékelhető ki, az mosml hibát jelez. Ha az mosml ezután a `val v1 = 4` deklarációt értékeli ki, akkor a táblába a `v1` név mellé a 4 értéket írja be, és a *továbbiakban* a `v1`-et a 4 szinonimájaként használja. Fontos, hogy a `v1` korábbi előfordulásait a változás nem érinti, a `v1`-et tartalmazó korábbi kifejezésekbe ugyanis a `v1` korábbi értéke (azaz a `v2` akkori értéke) már „beépült”.

Az SML-ben is vannak frissíthető változók, mert az SML nem tisztán funkcionális nyelv.

Mivel a *Deklaratív programozás* tárgy a deklaratív, azaz a logikai és a funkcionális programozási stílussal szeretné „megfertőzni” a hallgatókat, az SML imperatív ficamairól az előadásokon inkább szemérmesen hallgatunk. A kérdésre a választ mégsem akarom elsumákolni, ezért röviden összefoglalom a dolgot. Akinek ez nem elég, az mosml-lel együtt letöltött és az mosml-honlapon is megtalálható `General.sig`-ben és a *Moscow ML Language Overview*-ban utánanézhet.

Frissíthető változót az mosml-ben úgy hozhatunk létre, hogy a deklarációban a `ref` kulcsszóval rögtön azt is megadjuk, hogy az adott névnek mi lesz a kezdőértéke és ebből következően a típusa. Példa:

```
val frissitheto = ref 19;
```

A frissíthető változó értékét a `!`-lel jelölt ún. *dereferencing* vagy röviden *deref* függvény alkalmazásával csalhatjuk elő a változóból, ha a nevére alkalmazzuk. Példák:

```
!frissitheto;
20 - 1 = !frissitheto;
```

A frissíthető változónak új értéket a `:=`-vel jelölt értékadás-művelettel (infix operátorral) adhatunk. Példa:

```
frissitheto := 20;
frissitheto := !frissitheto - 1;
```

A kérdező folytatja: Az egyik évfolyamtársam említette, hogy a `:=` csak az értéket írja át, a mutatót nem. Mi történik a következő esetekben?

```
val v1 = 1
val v2 = v1
val v1 := 2
```

Oktató: Azt hiszem, a fenti részletes magyarázat után már mindenki tudja a választ. Azt is, hogy a harmadik sorban szintaktikai hiba van, mert a `v1` nem frissíthető változó.

13. fejezet

Részlegesen alkalmazható függvény, magasabbrendű függvény

13.1. Kérdés. Az alábbi függvény miért nem alkalmazható parciálisan (részlegesen):

```
fun proba a b = a + a
```

A típusa `fn: int -> 'a -> int`, tehát pl.

```
- proba 3
```

adhatna egy olyan függvényt, amely tetszőleges típusra `int`-et ad, ehelyett hibaüzenetet kapok.

Ugyanez a helyzet pl.

```
fun proba1 a b = 0
```

esetén is, pedig itt tényleg teljesen mindegy, hogy mik a típusok, az eredményt ez nem befolyásolja.

13.1. Válasz. ***Oktató:*** A kérdésére a választ a *Polimorfizmus* c. fejezetben találja meg.

14. fejezet

Típus, típuslevezetés, típusoperátor

14.1. Kérdés. A `->` típusoperátor merre köt? Jobbra vagy balra?

14.1. Válasz. ***Oktató:*** Jobbra köt.

14.2. Kérdés. Egy kérdésem lenne a zéhá 6.c feladatáról: a feladat a `fun f x y = y (x y)` típusának a levezetése.

Én úgy gondolkoztam, hogy mivel az `x` függvény argumentumul kapja `y`-t, amely szintén függvény, ezért `('a -> 'b) -> 'a` lesz az `x` típusa, eddig a helyes megoldás is ilyen. De ezután ebből egy olyan függvény lesz, amely `'b` típust ad eredményül (az `y` miatt), de az argumentuma egy függvény (`x y`), és ezért `('b -> 'a ) -> 'b`.

Tehát szerintem az egész : `(('a -> 'b) -> 'a) -> ('b -> 'a) -> 'b`, ami nyilván nem jó, mert begépeltem az ML értelmezőnek is... :(

```
val ('a, 'b) f = fn : (('a -> 'b) -> 'a) -> ('a -> 'b) -> 'b
```

Hol tévedtem?

14.2. Válasz. ***Oktató:*** A függvény törzsében (az egyenlőségjel utáni részben) az `y` argumentuma *nem* az `x` függvény, hanem az `x` függvény `y`-ra való *alkalmazásának* az *eredménye*, amit `'b`-val jelölt. Éppen innen kellett volna elindulnia: az `y`-t alkalmazzuk valamire (jelöljük a típusát `'a`-val), az eredményét pedig `'b`-val: `y : 'a -> 'b`. Ezután az `x`-et az `y`-ra alkalmazzuk, az eredménye pedig ugyancsak az `y`-nak (egy másik példányának) lesz az argumentuma, tehát az `x` eredményének és az `y` argumentumának azonos típusúaknak kell lenniük: `x: ('a -> 'b) -> 'a`. Összeállt minden, már csak be kell írni a dolgokat (a bal oldal alapján, figyelembe véve, hogy az egyenlőségjel mindkét oldalán azonos típusú értékeknek kell lenniük):

```
f: (('a -> 'b) -> 'a) -> ('a -> 'b) -> 'b
```

Figyelem: a zárójelek most fontosak! Az mosml válasza is ezt tükrözi.

14.3. Kérdés. Ha jól gondolom, egy SML típuskifejezésben a `*` típusoperátor a `->` típusoperátornál erősebben és ráadásul balra köt. Jól gondolom?

14.3. Válasz. ***Oktató:*** Abban igaza van, hogy erősebben köt, és másképp, mint a *jobbra kötő* `->` típusoperátor. De abban téved, hogy balra kötne – ugyanis se nem jobbra, se nem balra nem köt. Ezt illusztrálják az alábbi példák:

```
(1,2,3) : (int * int) * int;
(1,2,3) : int * (int * int);
```

amelyeket az mosml hibásnak talál:

```
! (1,2,3) : (int * int) * int;
! ^^^^^^^
! Type clash: expression of type
!   int * int * int
! cannot have type
!   (int * int) * int
- ! Toplevel input:
! (1,2,3) : int * (int * int);
! ^^^^^^^
! Type clash: expression of type
!   int * int * int
! cannot have type
!   int * (int * int)
```

Ha a `*` balra vagy jobbra kötne, a kettő közül az egyik zárójelezést jónak tartaná.

Persze nyilvánvaló módon nem fogadhatja el az mosml egyiket sem, ui. az `(int * int) * int` egy olyan pár típusa, amelynek az *első tagja* maga is egy pár, az `int * (int * int)` pedig egy olyan pár típusa, amelynek a `második tagja` maga is egy pár. Ezzel szemben az `int * int * int` egy *hármas* típusa.

14.4. Kérdés. Valaki árulja el nekem, legyen oly szíves, hogy hogyan kell levezetni az alábbi függvény típusát.

```
fun f p q r s = p(q r s);
```

14.4. Válasz. ***Oktató:*** No, akkor nézzük! Amint látjuk, az `r` és az `s` típusáról semmi nem derül ki, ezért jelöljük a típusokat egy-egy típusváltozóval:

```
r = 'a
s = 'b
```

A `q` függvény egy részlegesen alkalmazható függvény `r` és `s` argumentumokkal. Valamit varázsol, és visszaad valamilyen típusú értéket; mivel erről sem tudhatjuk, hogy milyen típusú, jelöljük `'c`-vel. Így a `q` függvény típusa:

```
q = 'a->'b->'c
```

A p-ről az látszik, hogy egy paramétere van, méghozzá a `q r s` függvényalkalmazásnak, vagyis a q függvény két paraméterrel történő meghívásának az eredménye. A q-ról tudjuk, hogy `'c` típusú értéket ad eredményül, így a p függvény értelmezési tartományát a `'c` típusú értékek alkotják. Az, hogy ebből a p milyen típusú eredményt állít elő, megintcsak nem derül ki, jeöljük hát `'d`-vel. Így:

```
p = 'c -> 'd
```

Most már, hogy a p, q, r és s típusát is meghatároztuk, összerakhatjuk az f függvény típusát:

```
f  = ('c -> 'd) -> ('a -> 'b -> 'c) -> 'a -> 'b -> 'd
```

Még egyszer, rövidebben: Mivel az s és az r argumentumok tetszőleges típusúak lehetnek, legyen az `r 'a`, az `s 'b` típusú. A q függvény, mégpedig olyan, amely r-re alkalmazva függvényt ad eredményül, amely azután s-ből állít elő valamit, legyen ennek típusa `'c`. Tehát `q: 'a -> ('b -> 'c)`, de a zárójel elhagyható, mivel a `->` jobbra köt. A p olyan függvény, amely a `q r s` értékéből állít elő valamit, legyen a típusa `'d`, tehát `p: 'c -> 'd`. Az f olyan függvény, amely p-re alkalmazva olyan függvényt ad, amely q-ra alkalmazva olyan függvény ad, amely r-re alkalmazva s-ből állít elő `'d` típusú értéket. Tehát f típusa:

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

és a felesleges zárójelek elhagyása után:

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

Egyszerű, nem?

A kérdező folytatja: Oké, tényleg egyszerű, csak azt honnan lehet tudni, hogy mind az s, mind az r paraméter (argumentum)? r miért nem részlegesen alkalmazható függvény? Ha pl. `fun f x y z = y x z`, akkor ebben mind az x, mind a z egyszerűen paraméter, és az x nem részlegesen alkalmazható függvény?

Oktató: Onnan tudható, hogy melyik név áll függvényalkalmazási-pozícióban, és melyik nem. A függvényalkalmazás a lehető legerősebben kötő művelet, a kiértékelés (hacsak zárójelek vagy precedenciák nem módosítják) mindig balról jobbra halad. A függvény nevét az argumentumtól egy vagy több szeparátor (formázó karakter vagy kerek nyitó zárójel) választja el. (A vessző, pontosvessző nem szeparátor!)

Nézzük még egyszer a kérdéses függvényt:

```
fun f p q r s = p(q r s);
```

Az érthetőség kedvéért tegyük ki a zárójeleket a deklaráció *mindkét* oldalán (az alábbi deklarációt az mosml nem tudná kiértékelni, mert a nyelv szintaxisa nem engedi meg, hogy a deklaráció bal oldalán a deklarálandó nevet zárójelbe tegyük):

```
fun (((f p) q) r) s = (p ((q r) s))
```

Tudjuk, hogy minden függvény egyargumentumú az SML-ben. Függvény az, amelynek van argumentuma. A fenti deklarációban tehát a következő dolgok jelölnek függvényt:

- `f p q r`, argumentuma: `s`
- `f p q`, argumentuma: `r`
- `f p`, argumentuma: `q`
- `f`, argumentuma: `p`
- `q r`, argumentuma: `s`
- `q`, argumentuma: `r`
- `p`, argumentuma: `(q r) s` !!!

A nevek közül tehát `f`, `q` és `p` jelölnek függvényt (pontosabban róluk *tudjuk*, hogy függvényt *kell* jelölniük), a többi nem függvényt jelöl (pontosabban róluk nem tudjuk, hogy milyen értéket jelölnek, akár függvényértéket is jelölhetnek). Világos?

14.5. Kérdés. Valaki legyen szíves és segítsen nekem megérteni, hogy a

```
fun f v = v o map
```

típusa miért az, ami.

14.5. Válasz. ***Oktató:*** Az `o` infix operátor a függvénykompozíció műveleti jele az SML-ben: jobb oldali `'d -> 'e` és bal oldali `'e -> 'g` típusú operandusából állít elő egy `'d -> 'g` típusú függvényt. (A `'d`, `'e`, `'g` jelöléseket addig használom, amíg nem tudok meg róluk többet.) A jelen esetben a szereposztás az, hogy a `map` van az `o` jobb oldalán, tehát neki kell `d' -> 'e` típusúnak lennie (az `map` eredménye a `v` paramétere). A `map` típusa `('a -> 'b) -> ('a list -> 'b list)`, ezért

```
'd = 'a -> 'b
'e = 'a list -> 'b list
```

A `'g`-ről nem tudunk semmi közelebbit. `v` típusa `('e -> 'g)`, behelyettesítve `('a list -> 'b list) -> 'g`, `v o map` típusa pedig `'d -> 'g`, behelyettesítve `('a -> 'b)->'g`. Így az `f` típusa:

```
(('a list -> 'b list) -> 'g) -> (('a -> 'b)->'g)
```

A felesleges zárójeleket elhagyhatjuk, és `g`-t a megszokottabb `'c`-re cserélhetjük:

```
(('a list -> 'b list) -> 'c) -> ('a -> 'b) -> 'c
```

14.6. Kérdés. A kérdés, amelyet Tóth kolléga helyezett a listára, a következő:

```
fun f a a b = a b
```

Mi ennek a szépségnek a típusa?

Nos, én kidolgoztam egy módszert a típuslevezetésre, de az itt nem működik. Miért nem? Nem tudom. :(Az mosml az alábbi válasszal áll elő:

```
'a -> ('b -> 'c) -> 'b -> 'c
```

Megpróbálom most visszafejteni a dolgokat. Tehát a jobb oldal `'c` típusú eredményt ad, a `b` típusára pedig azt mondja, hogy az simán `'b`. Ekkor az `a` típusa nem más, mint `'b -> 'c`.

Ezt kitűnően reprezentálja a zárójeles kifejezés. DE! Hát előtte is van egy `a`, annak miért nem a megállapított típusát írja ki az mosml??? Az én módszerem ezt adja:

```
('b -> a) -> ('b -> 'a)-> 'b -> 'a
```

A kérdés: miért nem ezt írja ki az mosml??? Még egy darab mókás mozzanatot engedek meg ezen a helyen: mosml barátunk szerint az alábbi két függvénynek ugyanaz a típusa:

```
fun f c a b = a b;
fun f a a b = a b;
```

Ez aztan megváltoztatja a szemléletemet! Azt kell mondjam, hogy az mosml valamit más-hogy csinál, mert amíg a

```
fun f c a b = a b;
```

deklarációnak nyilvánvaló a levezetése, és az alábbi eredményre vezet:

```
val ('a, 'b, 'c) f = fn : 'a -> ('b -> 'c) -> 'b -> 'c
```

addig a

```
fun f a a b = a b;
```

deklarációra nem ugyanazt a választ kellene adnia, pedig ugyanazt adja:

```
val ('a, 'b, 'c) f = fn : 'a -> ('b -> 'c) -> 'b -> 'c
```

Szerintem ezt még kell vesézni.

14.6. Válasz. ***Oktató:*** Ebben a dologban az a (szerintem csúnya) trükk, ahogyan az mosml értéket köt egy névhez. Mint ismeretes, egy névhez többször is lehet értéket kötni, és az mosml, amikor egy kifejezést kiértékel, nem a nevet, hanem a névhez *éppen akkor* kötött értéket építi be az eredménybe. Például:

```
- val a = 1;
> val a = 1 : int
- fun f () = a;
> val f = fn : unit -> int
- f ();
> val it = 1 : int
- val a = 2;
> val a = 2 : int
- f ();
> val it = 1 : int
```

Azaz az `f ()` eredménye nem változik meg attól, hogy az `a` név később már más értéket jelöl.

Mivel az SML-ben (a Prologgal ellentétben!) nincs egyesítés, csak behelyettesítés van, a bal oldalon kétszer is előforduló `a` argumentum valójában két, egymástól független argumentumot jelöl! Nézzünk egy nagyon egyszerű esetet és az mosml válaszát:

```
- fun f a a = 1;
> val ('a, 'b) f = fn : 'a -> 'b -> int
```

Ezért ekvivalens a `fun f a a b = a b` függvénydeklaráció típusa a `fun f x a b = a b` függvénydeklaráció típusával. Az utóbbi esetben az `f` típusának a levezetése feltehetőleg már senkinek nem jelent nehézséget.

Következő példaként nézzük az `f` két alkalmazását mosml alatt:

```
- fun f a a b = a b;
> val ('a, 'b, 'c) f = fn : 'a -> ('b -> 'c) -> 'b -> 'c
- f chr (fn x => x + 5) 3;
> val it = 8 : int
- f (fn x => x + 5) chr 3;
> val it = #"\^C" : char
```

Jól látszik, hogy az `f` első argumentumával a kutya se törődik. A helyzet fokozható: :-)

```
- fun f a a a a a b = a b;
> val ('a, 'b, 'c, 'd, 'e, 'f) f = fn :
  'a -> 'b -> 'c -> 'd -> ('e -> 'f) -> 'e -> 'f
```

Az SML-ben egy függvénynek nem lehetnének azonos nevű paraméterei! Tekintsük úgy, hogy ez egy hiba az mosml-ben. Az smlnj korrekt módon viselkedik:

```
Standard ML of New Jersey v110.42 [FLINT v1.5], Oct 16, 2002
- fun f a a b = a b;
stdIn:1.5-1.18 Error: duplicate variable in pattern(s): a
```

Haladóknak. Aki az eddigieket csak többé-kevésbé értette, már ne figyeljen ide! Érdekes módon az mosml szerint a `fun f a a b = a b` deklaráció típusa már *nem ekvivalens* a `fun f a x b = a b` deklaráció típusával, ugyanis az előbbié `'a -> ('b -> 'c) -> 'b -> 'c`, míg az utóbbié `('b -> 'c) -> 'a -> 'b -> 'c`. (Az, hogy az mosml az utóbbira nem betű szerint ezt írja ki, ne tévesszen meg senkit: a típusváltozók neve – feltéve, hogy az azonosak helyébe azonosakat írunk – tetszőleges más névre lecserélhető). Ennek oka a kifejezések kiértékelési sorrendjében keresendő.

14.7. Kérdés. Ha valakinek van kedve és ideje, leírná nekem, hogy hogyan kapjuk meg az `f` típusát az alábbi deklaráció esetén?

```
fun f (v, a) = map o (v a)
```

14.7. Válasz. ***Oktató:*** A feladat nem egyszerű. Szerncsére nemrég volt egy hasonló kérdés, ezért most a magyarázat lehet tömör. Az `o` a függvénykompozíció művelete, típusa `('e -> 'g) * ('d -> 'e) -> 'd -> 'g`.

A `map`, típusa `('a -> 'b) -> ('a list -> 'b list)`, az `o` bal oldalán van, ezért `('e -> 'g)` = `('a -> 'b) -> ('a list -> 'b list)`, és így `e'` = `('a -> 'b)` és `'g` = `('a list -> 'b list)`.

Az `o` jobb oldalán a `v a` van, ezért a típusának `'d -> 'e`-nek kell lennie. Az `a`-ra semmilyen megkötés nincs, jelöljük a típusát `'c`-vel. A `v` típusa tehát: `'c -> ('d -> 'e)`.

A megfelelő helyre behelyettesítve, amit csak lehet, kapjuk az `f` típusát:

```
f : ('c -> ('d -> 'e)) * 'c -> ('d -> 'g)
f : ('c -> ('d -> ('a -> 'b))) * 'c ->
                                  ('d -> ('a list -> 'b list))
```

A típusnevek szisztematikus cseréjével és a felesleges zárójelek elhagyásával azt kapjuk, amit az mosml is kiír (azért, hogy a régi és az új típusnevek ne keveredjenek, az új típusneveket nagybetűvel írjuk):

```
f: ('A -> 'B -> 'C -> 'D) * 'A -> 'B -> 'C list -> 'D list
```

14.8. Kérdés. Mi lesz a típusa? Miért??

```
fun f r s f = f(r, s)
```

Valami átírásféle... Volt ma az órán ilyesmi, de mi ennek a megoldása?

14.8. Válasz. ***Oktató:*** Először is jó tudni, hogy a két `f`-nek a `fun` és az `=` között – a függvénynévnek és az argumentumnévnek – semmi közük egymáshoz! Ha az argumentumot átnevezzük, a függvény jelentése nem változik meg. Első látásra nem tudható, hogy az `=` jobb oldalán az `f` vajon az argumentumként átadott függvény alkalmazását vagy az éppen most definiált függvény rekurzív alkalmazását jelenti-e. De ha jobban szemügyre veszük a definíciót, nyilvánvalóvá válik, hogy az utóbbiról nem lehet szó, mert a deklaráció két oldalán az `f` kétféleképpen lenne paraméterezve. Ezért az eredeti definíció a következővel ekvivalens:

```
fun f r s g = g(r, s)
```

Ettől kezdve már könnyű dolgunk van: `r` és `s` típusáról semmit nem tudunk, jelöljük ezeket `'a`-val, illetve `'b`-val. `g`-nek olyan függvénynek kell lennie – ez a definíció jobb oldalából tudható –, amely egy `'a *'b` típusú értékből egy `'c` típusú értéket ad eredményül. `'c`-ról ennél többet nem tudunk meg. `f`-et részlegesen alkalmazható függvény neveként deklaráltuk, ezért a típusa ez (megjelölve, hogy a típuskifejezés összetevői honnan származnak):

```
'a -> 'b -> ('a * 'b -> 'c) -> 'c
 |     |           |           |
 v     v    \------v------/     v
 r     s           g           f eredménye
```

14.9. Kérdés. Ha egy listából, amelynek az elemei listák, szeretnék egy olyan listát csinálni, amelynek az elemei az eredeti elem-listák elemei, akkor mi is a megoldás?

14.9. Válasz. ***Oktató:*** A `List.concat` függvény, amely például így definiálható:

```
fun concat lss = foldr op@ [] lss;
```

Haladóknak. Azt tanultuk, hogy a részlegesen alkalmazható függvények formális paraméterei jobbról balra haladva elhagyhatók a definíció mindkét oldalán. Az előző definíciónak ennek értelmében ekvivalensnek kell lennie a következővel:

```
val concat = foldr op@ [];
```

Az mosml próbálkozásunkat egy figyelmeztetéssel honorálja:

```
- val concat = foldr op@ [];
! Warning: Value polymorphism:
! Free type variable(s) at top level
!   in value identifier concat
> val concat = fn : 'a list list -> 'a list
```

Amikor először alkalmazzuk ezt a `concat` függvényt, az mosml újra figyelmeztet:

```
- concat [[1,2],[3,4,5]];
! Warning: the free type variable 'a
! has been instantiated to int
> val it = [1, 2, 3, 4, 5] : int list
```

Ettől kezdve a `concat` többé már nem polimorf:

```
- concat [[1.2],[3.4,5.0]];
! Toplevel input:
! concat [[1.2],[3.4,5.0]];
!         ^^^
! Type clash: expression of type
!   real
! cannot have type
!   int
```

De ha a `concat`-ot a korábbi definícióval hozzuk létre, a polimorf jellegét nem veszíti el:

```
- fun concat lss = foldr op@ [] lss;
> val 'b concat = fn : 'b list list -> 'b list

- concat [[1,2],[3,4,5]];
> val it = [1, 2, 3, 4, 5] : int list
- concat [[1.2],[3.4,5.0]];
> val it = [1.2, 3.4, 5.0] : real list
```

A magyarázatot az ún. értékpolimorfizmus adja meg, amelyről az FP-GYIK*Polimorfizmus* c. fejezetében valamivel többet, az ott hivatkozott leírásokban és a szakirodalomban sokkal többet olvashatnak az igazán haladók.

14.10. Kérdés. El tudná mondani valaki, hogy az alábbi kifejezésben mi az `x` típusa?

```
val x = let val xs = List.filter Char.isAlpha [#"1", #"3"]
        in xs
        end
```

14.10. Válasz. ***Oktató:*** `filter : ('a -> bool) -> 'a list -> 'a list`, azaz ha átad neki egy `('a -> bool)` típusú predikátumot (olyan függvényt, amelynek `bool` típusú az eredménye), kiválogatja az `'a list` típusú listából azokat az elemeket, amelyekre a predikátum igaz, és az elemek eredeti sorrendjét megőrző, ugyancsak `'a list` típusú listával tér vissza. A feladatban a predikátum `Char.isAlpha : char -> bool` és `[#"1",#"3"] : char list` a lista, amelyre `filter` a predikátumot alkalmazza. `Char.isAlpha` betűre igaz, egyébre hamis értékkel tér vissza: mivel az argumentumlistában nincsenek betűk, `xs` a `[] : char list` értéket kapja a `let`-kifejezésben, amelynek maga `xs` az eredménye. Végül a külső deklaráció az `x` a `let`-kifejezés eredményét, azaz `xs` értékét köti `x`-hez.

14.11. Kérdés. Meg tudná mondani valaki, hogy az alábbi kifejezésben mi az `x` típusa? Nekem nem megy. :(Úgy tűnik, hogy az `xs` bevezetése csak megtévesztés, hiszen a végén *x*-nek *xs* lesz az értéke!

```
val x = let val xs = List.find not [false, true] in xs end
```

A `List.find` eljárás a `false` értéket adja vissza, mert arra teljesül a `not` függvény, tehát az `xs` értéke (és ezáltal `x` is) `false`. Nem? Akkor most az `x` `bool` típusú?

14.11. Válasz. ***Hallgató:*** Az sml-jegyzet szerint (ld. a függelékében) az van, hogy a `List.find`, amikor olyan listaelemet talál, amelyre az alkalmazott – itt `not` – függvény igaz eredményt ad, akkor a `SOME v` (ahol `v` a talált elem) eredményt adja. Ha nem talált volna ilyet, akkor `NONE`-t ad vissza. Namost, `SOME` típusa `'a option`, tehát a válasz a feltett kérdésre: `x = false : bool option`.

Oktató: Az xs bevezetése valóban felesleges (nevezhetjük megtévesztésnek is), hiszen

```
let val xs = List.find not [false, true] in xs end =
                                    List.find not [false, true]
```

A hallgatói válasz helyes: `x` értéke `false`, a típusa pedig `bool option`, de a magyarázat nem elég pontos. — Az előadáson a `datatype` deklarációt és az `Option` könyvtárat tárgyaljuk, az alábbi válasz ezek ismeretében válik világossá.

`List.find` típusa és alkalmazásának eredménye, továbbá `SOME` és `NONE` típusa helyesen a következő.

```
List.find : ('a -> bool) -> 'a list -> 'a option
```

Ha a `List.find`-ot egy `'a -> bool` típusú `p` predikátumra alkalmazzuk, akkor olyan függvényt ad eredményül, amelyet egy `'a` típusú elemekből álló `xs` listára alkalmazhatunk. Ha a `List.find p xs` kifejezésben az `xs`-nek nincs a `p`-t kielégítő eleme (azaz olyan eleme, amelyre `p`-t alkalmazva `true` eredményt kapnánk), akkor `NONE` lesz a függvényalkalmazás eredménye. Ha az `xs`-nek van egy vagy több `p`-t kielégítő eleme, akkor - az `xs` balról számított első ilyen elemét `x`-szel jelölve - az eredmény `SOME x` lesz.

- a `SOME` ún. adatkonstruktorfüggvény, a típusa: `'a -> 'a option`,
- a `NONE` ún. adatkonstruktorállandó, a típusa: `'a option`.

14.12. Kérdés. Sziasztok, egy kis segítségre lenne szükségem. A kérdés: miért ez az `f` típusa?

```
fun f (x, y) = y(x+1, 2.0)
```
Megoldás: `int * (int * real -> 'a) -> 'a`

Szóval miért ez a megoldás? Meg egyébként is hogy kell az ilyet általánosságban megoldani?

14.12. Válasz. ***Oktató:*** Az `y`-t az `(x+1, 2.0)`-ra alkalmazzuk, ezért mivel az `x` `int` típusú (mert hozzá tudjuk adni az `int` típusú 1-et), a 2.0 pedig `real` típusú, az `y`-nak `int * real` az argumentuma, az eredménye pedig ismeretlen típusú, jelöljük `'a`-val. Az `y` alkalmazásának eredménye egyúttal az `f` alkalmazásának az eredménye, tehát az `f` típusa:

```
f : int * (int * real -> 'a) -> 'a
```

A kérdező folytatja: És ebben a feladatban miért ez az `f` típusa?

```
fun f x y z = x z y
```
Megoldás: `('a -> 'b -> 'c) -> 'b -> 'a -> 'c`

Oktató: A deklaráció jobb oldalából látható, hogy csak az `x` van függvényalkalmazási pozícióban, `z` és `y` a *részlegesen alkalmazható* `x` paraméterei. A paraméterek és az `x` eredményének típusát, mivel nem tudunk róluk többet, jelöljük típusváltozókkal:

```
z : 'a
y : 'b
x : 'a -> 'b -> 'c
```

A részekből már könnyű összerakni az egészt, figyelembe véve, hogy `f` is részlegesen alkalmazható függvény:

```
f : ('a -> 'b -> 'c) -> 'b -> 'a -> 'c
```

A kérdező folytatja: Már csak egy kérdésem maradt: mi az `f` típusa a következő definíció után?

```
fun f x y = y(x y)
```
Megoldás: `(('a -> 'b) -> 'a) -> ('a -> 'b) -> 'b`

Oktató: Nem győzöm elégszer hangsúlyozni, hogy a típusegyenletek megoldásakor mindig a deklaráció jobb oldalát kell először szemügyre venni. A legfontosabb, hogy megállapítsuk, melyik név van *függvényalkalmazási pozícióban.*

Ebben a feladatban mind `y` első előfordulása az `=` jobb oldalán, mind `x` függvényalkalmazási pozícióban van, az `x` függvény paramétere `y`, az `y` függvény paramétere pedig az `x y` függvényalkalmazás *eredménye*.

A típusokról semmi konkrétumot nem tudunk meg, ezért az `y` típusa a lehető legspecifikusabban így írható fel: `'a -> 'b`. Ha ez megvan, már rutinfeladat a levezetés:

```
x : ('a -> 'b) -> 'a
```

Az eredménynek `'a` típusúnak kell lennie, mert az `y`-t alkalmazzuk rá! Végül behelyettesítve és kirakva a zárójeleket, ahol szükséges:

```
f : (('a -> 'b) -> 'a) -> ('a -> 'b) -> 'b
```

14.13. Kérdés. Meg tudná valaki mondani, hogy ennek mi a típusa:

```
val x = map (foldl op<> (3<>4))
```

14.13. Válasz. ***Hallgató:*** A `foldl op<>` függvény egy `3<>4 = false` értékű, `bool` típusú egységelemmel kezdve szépen összehasonlítgatja egymás után egy lista elemeit (ezek is csak `bool` típusúak lehetnek, hogy működjön az `op<>`), ennek tehát egy lista az argumentuma, és `bool` típusú a végeredménye. Viszont ott van előtte a `map`, amely miatt az egész egy `bool list` típusú listákból álló listára alkalmazható, és a végeredménye a részlisták eredménye egy listába fűzve. Azaz: `bool list list -> bool list`.

Szóljatok, ha valami nem stimmel, vagy nem érthető…

Másik hallgató: Megfogalmazom másképpen, formálisabban, hátha valakinek segítek vele. A `3 <> 4 = false`, a típusa `bool`, ez az ún. *egységelem* (trükkös). Ebből következően a `op <>` `bool * bool -> bool` típusú ebben a feladatban. Tehát a listának, amelyre az `x`-et alkalmazzuk, `bool list`-nek kell lennie. Ezért

```
foldl op<> (3<>4) : bool list -> bool
```

A `map` típusa:

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

Itt most `'a` = `bool list` és `'b` = `bool`, és ezért

```
x : bool list list -> bool list
```

14.14. Kérdés. Írjon az alábbi fejkommentet kielégítő SML-függvényt! Segítségül példát is mutatunk az alkalmazására.

```
(* nthFmBehind(i, xs) = xs hátulról számított i-edik eleme,
        ahol egy lista utolsó elemének a sorszáma 1
        PRE: i > 0
   nthFmBehind : (int * 'a list) -> 'a
*)
```

Példa: `nthFmBehind (3, ["w","x","y","z"]) = "x"`

Ehhez a feladathoz kérek segítséget. A probléma az, hogy ha nem kezelem le az `xs = []` esetet, akkor nem enged tovább, ha lekezelem, akkor viszont a visszaadott érték nem megfelelő, hiszen nem tudom visszaadni a listabeli elem típusát. Mi a megoldás?

14.14. Válasz. ***Hallgató:*** Itt van egy megoldásom:

```
fun nthFmBehind (i, xs) =
      let
          val j = length xs - i
          fun s (0, (y::ys)) = y
            | s (n, (y::ys)) = s(n-1, ys)
      in
         s(j, xs)
      end;
```

Példa:

```
- nthFmBehind (3, ["w","x","y","z"]);
> val it = "x" : string
- nthFmBehind (5, ["w","x","y","z"]);
! Uncaught exception:
! Match
- nthFmBehind (1, []);
! Uncaught exception:
! Match
```

Oktató: A hallgató megoldása elfogadható, de nem szép benne, hogy az `s` segédfüggvény nem kezel minden esetet, amire az mosml figyelmeztet is.

```
! Toplevel input:
! ..............s (0, (y::ys)) = y
!               | s (n, (y::ys)) = s(n-1, ys)
! Warning: pattern matching is not exhaustive
```

A korrekt megoldásnak minden esetet kezelnie kell, az üres listára pedig kivételt (exception) kell jeleznie. Ez lehet a hasonló esetekben jelzett `Empty` (vö. `List.hd`, `List.tl`), esetleg a `Subscript` (vö. `List.nth`, `List.take`, `List.drop`), vagy egy erre a célra deklarált saját kivételnév. Egyébként kivételt jelez a hallgató megoldása is, de a `Match` hibaüzenet szerintem nem elég kifejező az adott esetben.

Gyorsan megírható az a változata, amely az `i = 0` és az `i > length xs` speciális eseteket is automatikusan kezeli, ha a `List.nth : 'a list * int -> 'a` és a `length : 'a list -> int` függvényeket alkalmazzuk. `List.nth` alkalmazásához tudni kell, hogy a lista fejének (bal szélső elemének) 0 az indexe.

```
fun nthFmBehind (i, xs) = List.nth(xs, length xs - i);
```

Példa:

```
- nthFmBehind (3, ["w","x","y","z"]);
> val it = "x" : string
- nthFmBehind (5, ["w","x","y","z"]);
! Uncaught exception:
! Subscript
- nthFmBehind (1, []);
! Uncaught exception:
! Subscript
```

Kifejezőbbek lesznek a hibaüzenetek, ha az üres listára is írunk egy klózt:

```
fun nthFmBehind (_, []) = raise Empty
  | nthFmBehind (i, xs) = List.nth(xs, length xs - i);
```

Példa:

```
- nthFmBehind (3, ["w","x","y","z"]);
> val it = "x" : string
- nthFmBehind (5, ["w","x","y","z"]);
! Uncaught exception:
! Subscript
- nthFmBehind (1, []);
! Uncaught exception:
! Empty
```

Ugyancsak gyorsan megírható az a változata, amely a `List.nth` mellett a `rev`-et alkalmazza.

```
fun nthFmBehind (_, []) = raise Empty
  | nthFmBehind (i, xs) = List.nth(rev xs, i-1);
```

Végül következzen egy olyan változata, amely a `List.nth` függvényt nem alkalmazza.

```
fun nthFmBehind (_, []) = raise Empty
  | nthFmBehind (i, xs) =
      let fun nFmB (0, z::zs) = z
            | nFmB (i, _::zs) = nFmB(i-1, zs)
            | nFmB (_, []) = raise Subscript
      in
          nFmB(i-1, rev xs)
      end;
```

A két utóbbi megoldás esetén az mosml válaszai az előző példában bemutatottakkal azonosak lesznek.

14.15. Monológ. ***Hallgató:*** Azért írom az info2000-re is, mert azt többen olvassák. Biztos sok hülyeséget is írok, és nem teljesen korrekt a nyelvezet, de remélem, mindenkinek hasznára lesz, legalábbis a zhig :)! Ha tetszik a dolog, tegyétek fel az infositera is! Típuslevezetéses feladatokat valahogy így lehet megoldani:

1. Vegyük az alábbi példát. Saját függvényt definiálunk.

   ```
   fun valami x = x;
   ```

Ennek típusa: `'a -> 'a`.

A `'a`, `'b`, `'c` stb. tetszőleges típust jelölnek, típusváltozóként foghatók fel, kb. úgy, mint a C++ban a template. A típuslevezetést úgy kapjuk meg, hogy a rendszer felteszi, hogy `x`-nek `'a` típusa van, ekkor viszont a válasz is `'a` típusú, hiszen pontosan `x`-et adja vissza a függvény. A `->` jel a leképzés jele, bal oldalán a függvény argumentumának, jobb oldalán a függvény által visszaadott értéknek a típusa latható. Ez a függvény tehát alkalmazható tetszőleges típusú paraméter esetén, és ugyanolyan típusú értéket ad vissza.

2. Vegyünk egy olyan példát, amelyben egy ennes szerepel:

   ```
   fun valami (x, y) = x;
   ```

   Ennek típusa: `'a * 'b -> 'a`

   A `*` jel a típuskifejezésben az enneseket jelöli (egyébként a *direktszorzat* jelét akarja sugallni), és erősebben köt, mint a `->`, tehát a kifejezés így értelmezendő:

   ```
   'a * 'b -> 'a
   ```

   Egy `'a` és egy `'b` típusú értékekből álló párt képezünk le egy `'a` típusú értékre.

3. Vegyünk egy olyan példát, amelyben egy lista szerepel:

   ```
   fun valami (fej::farok) = fej;
   ```

   Ennek a típusa: `'a list -> 'a`

   Itt a `list` típusoperátor az új: egy adott típus mögé írva az ilyen típusú elemekből képzett listát jelöli. Balra köt, így pl. `'a list list` egy `'a` típusú elemekből álló listák listáját jelöli.

4. Vegyünk egy részlegesen alkalmazható függvényt, például:

   ```
   fun valami x y = x;
   ```

   Nyilván itt `y` értéke tetszőleges, nem befolyasolja a függvény eredményét. A függvény az `x` paraméter értékét adja vissza, a típusa: `'a -> 'b -> 'a`. Mivel a `->` operátor jobbra köt, ez így értelmezendő:

   ```
   'a -> ('b -> 'a)
   ```

   Emlékezzünk vissza, hogy minden függvény csak egy paraméterű lehet, tehát a `valami` függvénynek itt csak az `x` a paramétere. Ez a `valami x` egy másik függvényt fog visszaadni, amelynek az `y` lesz a paramétere.

   Vagyis a `valami` fugggény paraméterként kap egy `'a` típusú értéket: ez lesz az `x`. Visszaad egy függvényt. Ez a visszaadott függvény paraméterként kap egy `'b` típusú értéket: méghozzá az `y`-t. Ennek visszatérési értéke az `x` érték lesz, amelyről ugye már tudjuk, hogy `'a` típusú.

   Hogy egy kicsit érthetőbb legyen a dolog, nézzük egy példát `valami x y`-ra, azaz a `valami` függvény „teljes” alkalmazására:

   ```
   valami 3 4
   ```

válaszul 3-at kapunk. Részlegesen alkalmazva a függvényt:

```
valami 3
```

válaszul egy függvényt kapunk, amelynek van 1 paramétere, de annak értékétől függetlenül 3-at ad vissza, tehát gyakorlatilag konstansfüggvény.

5. Eddig általánosságban beszéltünk, és célszerű egy feladat megoldásakor is először így általánosságban levezetni a típust. A konkrét típusokat, úgymint `int`, `real`, `bool`, `unit` stb. – ahol ezeket egyáltalán meg lehet állapítani – kétféleképpen lehet meghatározni:

   (a) `fun valami (a : real) = a;`

   A `:` és egy típusnév segítségével előírhatjuk egy meghatározatlan típusú kifejezés típusát. Itt `a` típusát `real`-re állítottuk. Fontos a zárójelezés, mivel a „típuscastolás” nagyon alacsony precedenciájú. Mindjárt meglátjuk, mi lenne zárójelek nélkül.

   A függvény típusa most: `real -> real`, ugyanis explicite előírtuk, hogy a paramétere `real` típusú legyen, és mivel a visszatérési értéke megegyezik az átadott paraméter értékével (identitásfüggvényről lévén szó), ezért nyilván az is `real` típusú.

   `fun valami a : real = a;`

   Itt most a `real` típust nem az `a` paraméterre, hanem a függvény eredményére írtuk elő! Az eredmény itt ettől függetlenül ugyanaz, de egyéb esetekben ez nem feltétlenül teljesül.

   (b) A beépített függvények mindegyikére megvan, hogy milyen típusokat vár el és ad vissza alapértelmezésként. A legtöbb aritmetikai függvény alapértelmezésben `int` típusokkal dolgozik, a `/` viszont `real`-lel. Többszörösen terhelt nevek esetén az alapértelmezéseket felül lehet bírálni, ha explicite megadjuk a típusokat, mint az előző pontban láttuk.

   Másrészt meghatározó a számkonstansok típusa is. Így pl.

   - `fun valami x = 2 * x` esetén `valami: int -> int`.
   - `fun valami x = 2.3 * x` esetén `valami: real -> real`.
   - `fun valami x = x/2` hibás, mivel a `/` operátor `real` operandusokat vár, a 2 viszont `int` típusú.
   - `fun valami x = x / 2.0` esetén `valami: real -> real`.
   - `fun valami x = x > 2` esetén `valami: int -> bool`.
   - `fun valami x = x > 2.0` esetén `valami: real -> bool`.

   (c) Fontos még a `''a`, `''b`, `''c` stb. alakú típusváltozók említése is. Ezek ugyanolyanok, mint a `'a`, azzal a különbséggel, hogy a `''` azt jelzi: a típuson elvégezhetőnek kell lennie az egyenlőségvizsgálatnak. Pl. két `int` egyenlősége vizsgálható, tehát a `''a` konkrétan lehet `int`, viszont nem lehet pl. `real -> int`, mert függvények egyenlőségét nem lehet megvizsgálni.

   `fun valami (x,y) = x = y;`

   A típusa: `''a * ''a -> bool`. A `bool` visszatérési érték nyilvánvalóan az `=` operátorból adódik. A `''a` jelöli, hogy a konkrét típusoknak egyenlőségvizsgálatot megengedő típusoknak kell lenniük. Megfigyelhető még az is, hogy mindkét

paraméter `''a` típusú. Ez azért van, mert `''a * ''b -> bool` típus esetén pl. megpróbálhatnánk összehasonlítani egy `string`-et egy `int`-tel, aminek nyilván kevés értelme lenne!

14.16. Kérdés. Mit tegyek, ha az sml nem enged meg olyan dolgokat, mint a Prolog, például önmagát tartalmazó listákat... Konkrétabban az alábbi Prolog-eljárást szeretném megvalósítani sml-ben:

```
lls(0, []) :- !.
lls(N, [L|L]):- N1 is N-1, lls(N1, L).
```

Ha valakinek sikerül, légyszi szóljon!

14.16. Válasz. ***Oktató:*** Az SML szigorú típusossága miatt ez nem sikerülhet. Más utat kell keresnie.

14.17. Kérdés. Arra lennék kíváncsi, hogy ennek mi a típusa?

```
val x = fn (z, y, f) => f y z

x : 'a * 'b * ('b -> 'a -> 'c) -> 'c
```

az mosml szerint, de ez hogyan jön ki? Mert ha nézem a jobb oldalt, akkor

```
z = 'a
y = 'a -> 'b
f = 'a -> 'b -> 'c
```

és akkor `'a * 'b * 'c`-nek kellene lennie ott, nem? (Úgy láccik, hogy nem, de miért?)

14.17. Válasz. ***Hallgató:*** Vigyázz, az `y` nem függvény! Így kellett volna felírnod:

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

amiből névcserével megkapod az mosml válaszát.

A kérdező folytatja: Amire még kíváncsi lennék:

```
val x = fn p => fn (q, r) => p q r
```

Hallgató: Szerintem az előzőhöz hasonló fejtegetéssel:

```
q : 'a
r : 'b
p : 'a -> 'b -> 'c
x : ('a -> 'b -> 'c) -> 'a * 'b -> 'c
```

14.18. Kérdés. Kisegítene valaki, hogy mi az `f` típusa és miért:

```
fun f g (h,i) = g (i,h) i
```

14.18. Válasz. ***Oktató:*** Annyit magyaráztuk már ezeket a típusegyenleteket! Ha még mindig nem érti, olvassa el a <http://dp.iit.bme.hu/sml/eloadas_anyagok/fpTiplev/tipuslevezetes.html> <http://dp.iit.bme.hu/sml/eloadas_anyagok/tipuslevezetes.pdf> segédletet, abból talán megérti, hogy az mosml szerint az `f` típusa miért éppen ez:

```
('a * 'b -> 'a -> 'c) -> 'b * 'a -> 'c
```

14.19. Kérdés. Jó szokásomhoz híven :) akadt megint egy-két dolog, ami nem teljesen világos. Kérem, aki tud, segítsen, mert fogy az idő…

```
- val v2 = app (print o str);
> val v2 = fn : char list -> unit
```

Szóval ez az eredmény hogyan jön ki? Elsősorban az `app` függvényre vagyok kíváncsi, de az egész megoldás menetét sem vetném meg. :)

14.19. Válasz. ***Oktató:*** Nézzük az alkalmazott függvények típusát:

```
str   : char -> string
print : string -> unit
o     : ('a -> 'b) * ('c -> 'a) -> 'c -> 'b
```
infix!

A behelyettesítések:

```
('c -> 'a) <--- (char -> string)
```
és
```
('a -> 'b) <--- (string -> unit)
```
, azaz
```
'a <--- string,
'b <--- unit,
'c <--- char
```

A `print o str` kifejezés eredménye egy függvény, az `str` és a `print` kompozíciója. A behelyettesítések után a típusa: `char -> unit`. Erre a függvényre alkalmazzuk `app`-ot, amelynek az első paramétere van csak lekötve.

```
app : ('a -> unit) -> 'a list -> unit
```

A behelyettesítések:

```
('a -> unit) <--- (char -> unit), azaz
'a <--- char.
```

`app (print o str)` = `v2` típusa az `'a <-- char` behelyettesítéssel: `char list -> unit`. (A `v2` egy karakterlista elemeit írja ki a standard outputra.)

A kérdező folytatja: A kérdésem ugyanaz, mint előbb...

```
- val v4 = map (String.fields Char.isPunct);
> val v4 = fn : string list -> string list list
```

Oktató: A válasz is ugyanaz: el kell végezni a behelyettesítéseket a megfelelő sorrendben!

```
String.fields : (char -> bool) -> string -> string list
Char.isPunct  : char -> bool
(String.fields Char.isPunct) : string -> string list
map : ('a -> 'b) -> 'a list -> 'b list

('a -> 'b) <--- (string -> string list), tehát
'a <--- string,
'b <--- string list
```

Továbbá

```
(map f) típusa: 'a list -> 'b list
```

ui. első paramétere le van kötve az `f : 'a -> 'b` függvénnyel! Azaz

```
map (String.fields Char.isPunct) == v4 típusa tehát a
```

behelyettesítések után

```
string list -> string list list
```

A kérdező folytatja: A következő feladatban sajnos még az eredmény sincs meg, mert nem tudtam rájönni, hogyan lehetne beinklúdolni a `Math` struktúrát. `use "Math.uo"`, `use "Math.ui"`, `use "Math.sig"`? Egyik se volt elég jó... :)

```
val v5 = foldr op:: [Math.sin];
```

Hallgató: Nem inklúdolni, hanem lódolni kell: `load "Math"`. Úgy emlékszem, ez még a diákon is szerepel, nemcsak az mosml manualban.

Oktató: Nem lesz meglepő, amit írok: el kell végezni a behelyettesítéseket...

```
op::       : 'a * 'a list -> 'a list
[Math.sin] : (real -> real) list
foldr      : ('A * 'B -> 'B) -> 'B -> 'A list -> 'B
```

Kisbetűk helyett nagybetűket használtam, hogy különbözzenek `op::` típusváltozóitól.

`('A * 'B -> 'B) <--- 'a * 'a list -> 'a list,`
`'B <--- (real -> real) list`, azaz
`'A <--- 'a,`
`'B <--- 'a list == (real -> real) list,`
`'A == 'a <--- (real -> real)`

```
foldr op:: [Math.sin] == v4
```

típusa (a `foldr` első két argumentuma van lekötve):

`v4: 'A list -> 'B`, és a behelyettesítés után
`(real -> real) list -> (real -> real) list`

15. fejezet

Polimorfizmus

15.1. Kérdés. Első próbálkozás:

```
- fun b x = x+1;
> val b = fn : int -> int
```

Második próbálkozás:

```
- fun b x = x;
> val 'a b = fn : 'a -> 'a
    ^^^^
```

Ide miért jön be az `'a`? Vagy inkább úgy kérdezem, van-e lényegbeli különbség az `'a`-s és az `'a` nélküli típusnál?

A könyv (4. javított kiadás) 53. oldalán `fun id x = x`-re `val id = 'a -> 'a` van írva válasznak, ellenben ha beírom mosml-be, akkor ott is megjelenik az `'a` (ugyanígy a megfelelő fólián is ki van írva az `"a`).

15.1. Válasz. ***Hallgató:*** Az első próbálkozással szemben a második próbálkozás polimorf függvényt definiál, az okot tehát a polimorfizmus kezelésében kell keresni.

Az `'a-t` szerintem azért írja ki, hogy mutassa, hogy `b` egy polimorf (típus-paraméteres) dolog. Mivel ez a típusából amúgy is látszik, én ezt nem tekintem lényegi különbségnek (a `val b = fn : 'a -> 'a` felíráshoz képest természetesen).

A könyvben valószínűleg azért van másképp, mert az írásakor használt mosml verzió még nem akarta külön is hangsúlyozni, hogy egy függvény polimorf.

Oktató: A függvény definiálásakor – a paraméteres `datatype`-deklarációhoz hasonlóan – megadhatunk típusparamétereket is, pl.

```
fun ('a, 'b) f (x, y) = ((x :'a) , (y :'b));
val ('a, 'b) f = fn : 'a * 'b -> 'a * 'b
```

ami fokozott típusellenőrzést tesz lehetővé. Amíg a következő függvénydefiníció símán lefordul:

```
fun f (x,ys) = x::ys;
val 'a f = fn : 'a * 'a list -> 'a list
```

addig ugyanennek `'a`-val és `'b`-val paraméterezett változatát az mosml hibásnak mondja, mert a programozó típusparaméterekkel kifejezett *szándéka* és a függvény *definíciója* között ellenmondást lát:

```
! Toplevel input:
! fun ('a, 'b) f (x :'a , ys : 'b) = x::ys;
!                                     ^^
! Type clash: expression of explicit type 'b
! cannot have type
!   'a list
```

A válaszoló jól tippelt, az mosml értelmező a jegyzet írásakor még másképpen működött.

15.2. Kérdés. Az egyik SML gyakorlófeladat a `List.length` újraírása. Ezt nem fogadta el a gyakorlórendszer: `val len = foldl (fn (a,b) => b+1) 0`.

Ha az `a` változó típusát leszűkítem, vagy `fun` alakban írom (`fun len xs = foldl (fn (a,b) => b+1) 0`), akkor minden OK.

A kérdéseim:

1. Miért nem működik az eredeti?
2. Melyek azok a kifejezések, amelyeknél a `fun a x = f x` alakot nem lehet átírni az `val a = f` alakba?

15.2. Válasz. ***Hallgató:*** Sikerült rátalálnod a típusos funkcionális nyelvek egyik gyengéjére. A gond az, hogy a `len` polimorf függvény, ráadásul pont a ki nem írt argumentumának a típusa polimorf. Ezt sajnos a típusrendszer (bizonyos ismert, sok helyen leírt okok miatt) *toplevel* szinten nem tudja elfogadni. Ha kiírod az argumentumot (akár `fun` kulcsszóval, akár *lambda* alakban), akkor feloldod a problémát.

Ha többet szeretnél megtudni az okáról, a neten számos ezzel foglalkozó cikk, leirás található.

Oktató: A jelenség megértéséhez meg kell ismerni az expanzív és a nemexpanzív kifejezés fogalmát, és azt, hogy hogyan értelmezi ezeket az SML nyelv, és hogyan kezelik az egyes SML-értelmezők.

A polimorfizmusról és a kifejezések expanziójáról egy 8+1 oldalas magyar nyelvű összefoglalót felraktam a tárgy honlapjára:

- <http://dp.iit.bme.hu/sml/eloadas_anyagok/dp03s-polimExpan_p1.pdf> (1 dia/lap)
- <http://dp.iit.bme.hu/sml/eloadas_anyagok/dp03s-polimExpan_p2.pdf> (2 dia/lap)
- <http://dp.iit.bme.hu/sml/eloadas_anyagok/dp03s-polimExpan_p1.ps.gz> (1 dia/lap)

A *Moscow ML Owner's Manual* „Value polymorphism" c. 12. fejezete ugyanerről angolul ad rövid áttekintést (lásd pl. <http://dp.iit.bme.hu/download.html>, *Moscow ML Owner's Manual*).

Akit még bővebben érdekel, keressen „value polymorphism" témájú cikkeket a weben.

16. fejezet

Modul (struktúra, szignatúra)

16.1. Kérdés. Bizonyára sok emberrel megtörtént, hogy az otthon működő SML progija a beadáskor valamilyen inkompatíbilitási hibára hivatkozva le sem fordult.

16.1. Válasz. ***Oktató:*** Inkompatíbilitásra akkor szokott panaszkodni az mosml, ha egy program egyes moduljait egy adott környezetben, más moduljait egy másik környezetben fordították le. Az adott esetben ez csak akkor lehetséges, ha (1) a forráskód (.sml) mellett a tárgykódot (.uo) is elküldte, *és* (2) a modul lefordítása a benti gépen valamilyen szintaktikai hiba miatt meghiúsult. Ilyenkor az otthon lefordított modult próbálja meg összeszerkeszteni az mosml a benti gépen előre lefordított modulokkal.

A kérdező folytatja: Nos: nekem sikerült újra működésre bírnom (a magyarázatát nem tudom, mázlimra egy régi működött, és a különbséget vizsgáltam meg). Írjátok be az első sorok közé: `open TCikcakk`.

Oktató: Igen, azt történt, amit leírtam: a `Cikcakk` modult nem tudta lefordítani. Amit csinált, az az egyik lehetséges megoldás. Megoldás az is, ha a `TCikcakk` modulban a

```
datatype dir = n | e | s | w | ne | nw | se | sw
```

deklarációval létrehozott összes adatkonstruktorra, ahol csak használja őket, a teljes nevükkel hivatkozik, pl. `TCikcakk.nw`.

A kérdező folytatja: Betettem a `Cikcakk` modulba `local ... in ... end` „zárójelek” közé, hogy szebb legyen. Otthon jó volt... Egy hasonló tartalmú, csak `let`-tel próbálkozó modulom fordítása meghiúsult... Ez van.

Oktató: A titok nyitja itt van! A modulok szintaxisáról nem nagyon beszéltünk az órákon, de aki megnézi a *Moscow ML Language Overview Grammar for the Moscow ML module language* c. fejezetét, láthatja, hogy a modulban csak deklarációk lehetnek, ezért csak `local`-deklaráció használható, `let`-kifejezés nem. A megoldás tehát egy ilyen szerkezet lehet:

```
structure x =
struct
  local
```

```
        open Math
    in
        val s = sin
    end
end
```

A kérdező folytatja: Ezt nem egészen értem, meg van tiltva az alábbi eset?

```
fun valami x =
      let
         open TCikcakk
      in
         x+2
      end
```

Elvileg az `open` csak `val TCikcakk.nw` típusú dolgokat tesz lehetővé, akkor ez miért baj? Másrészt otthon lefordult...

Oktató: Nem, ez nincs letiltva, hiszen itt egy *függvény belsejében*, egy lokális kifejezésben nyitja meg és használja a `TCikcakk`-ot. De ha a `let`-kifejezést egy *struktúra* (modul) törzsébe – a `struct` és az `end` közötti részbe – írja be, ahová csak deklarációkat lehet írni, akkor baj van, az hiba. Ugyanis bármilyen kifejezés, így bármilyen `let`-kifejezés is, csak egy deklaráció jobb oldalán használható.

Joggal kérezheti, hogy akkor az mosml *értelmező* miért hajlandó kiértékelni tetszőleges kifejezést, miért nem csak deklarációkat fogad el? Barátságos gesztusként, azért, hogy kényelmesebbé tegye az interaktív használatot! Ui. minden *kifejezés elé* odaképzeli a `val it =` deklarációkezdetet.

Ha tehát egy tetszőleges deklarációban, pl. egy `local-deklarációban` egy `open`-nel megnyit egy struktúrát (pl. a `TCikcakk`-ot), akkor az adott deklaráció *érvényességi körében* (scope) használhatja a megnyitott struktúrában deklarált nevek rövid változatát is (pl. az `nw`-t). Baj ebből csak akkor van, ha ugyanaz a név egynél több modulban van deklarálva, – azaz többszörösen van terhelve – és a rövid változatukat *egyszerre* szeretné használni. Ezt a *láthatósági szabályok* (visibility rules) nem engedik meg: egy később láthatóvá tett vagy deklarált név eltakarja a korábban láthatóvá tett vagy deklarált azonos alakú neveket.

16.2. Kérdés. A házit fogadó gép melyik `TCikcakk`-ot használja? Mert nekem szuksegem van egy `x` irányra belső adminisztrációs felhasználásra, ezért bele kellett nyúlnom, hogy azt hozzárakjam.

16.2. Válasz. ***Oktató:*** Finoman szólva nem javaslom, hogy módosítsa a `TCikcakk` modult! A specifikacio azért van, hogy betartsák. A teszteléskor csak a kiírásban megadott, ill. a keretprogrammal együtt letölthető specifikációt `TCikcakk.sig`, `Cikcakk.sml`) fogadja el a számítógép.

Képzelje el, mi lenne abból, ha ugyanezt akarná tenni egy igazán nagy rendszerben, ahol 13 másik programfejlesztő közösen dolgozik egy nagy projekten! Keressen más megoldást!

16.3. Kérdés. Hogy lehet az sml-ben elérni azt, hogy a kimenet ilyesmi legyen: `[[e,s,n,n,n]]`? Talán rosszul adtam meg az `irany` típust? Ugyanis a kimenet egyelőre ilyen alakú: `[[?e, ?s, ?n, ?n, ?n]]`.

16.3. Válasz. ***Hallgató:*** Úgy, hogy nem a rövid `n`, `e`, `w`, `s` neveket használod, hanem a hosszú `TSatrak.n`, `TSatrak.w` stb. neveket. A kimenet ekkor ilyen lesz: `[[TSatrak.n,TSatrak.w,...]` de ez nem baj, ez így jó.

A kérdező folytatja: Én `TSatrak.n`-et használok, de ettől függetlenül a fenti kérdőjeles megoldást kapom...

Oktató: Nézze meg az „egysátras” mintamegoldást a <http://dp.iit.bme.hu/dp01s/egysator> címen a honlapon. Semmiféleképpen *ne* legyen az `irany` adattípus más struktúrában definiálva, mint a `TSatrak.sml`-ben.

17. fejezet

SML Basis Library

17.1. Kérdés. Azt szeretném megkérdezni, hogy mi az az `option`, ill. a `unit` típus? és ehhez kapcsolódva, hogy mit is csinál pontosan az `isSome`, `getOpt`, `valOf`, `getItem` és `mapPartial`? Nem igazán tudtam rájönni ezekre a jegyzetből. Egyáltalán kellenek ezek a dolgok?

17.1. Válasz. ***Oktató:*** Kellenek. Az előadásokon többször elmagyaráztam a jelentésüket. Az előadásdiákon, az *SML Basis Library*-ben az `Option` és a `General` modulokban, továbbá a GYIK jelen fejezetében ezekről a dolgokról elég részletes magyarázatok olvashatók.

17.2. Kérdés. Az sml jegyzetben a `Structure Option` [az `Option` modul] leírásában van ez a `SOME`, de fogalmam sincs, hogy mit jelent. Ha valaki meg tudná mondani, igen sokat segítene (csak a motiváció kedvéért).

17.2. Válasz. ***Hallgató:*** Van úgy, hogy azt szeretnénk, hogy egy azonosító értékül felvehesse azt is, hogy semmi. Pl. írunk egy 'lista első eleme' függvényt, és azt szeretnénk, hogy a függvényünk az üres listára azt mondhassa, hogy nincs eredmény. Kb. erre való az `'a option` típus.

Amikor az `'a option` típust használva azt akarjuk értékül adni, hogy semmi, akkor a `NONE` adatkonstruktorállandót kell használnunk. Ha pedig valódi értéket akarunk értékül adni, akkor a `SOME` adatkonstruktorfüggvényt.

Az `'a option` deklarációja ez:

```
datatype 'a option = SOME of 'a | NONE;
```

Tehát vagy hordoz valamilyen értéket, ekkor `SOME ertek` alakú; vagy nem, ekkor `NONE` alakú. Példa:

```
fun listafeje (x::xs) = SOME x
   |listafeje [] = NONE;
```

Kis kényelmetlenség, hogy minden esetben mintaillesztést (vagy ezt elvégző függvényhívást) kell használnunk, hogy a hordozott értékhez hozzáférjünk, már ha egyáltalán van hordozott érték.

Szerencsére vannak könyvtári függvények (pl. `List.mapPartial`), amelyek „eltakarják” ezt a komplexitást.

Remélem, megvilágítottam a dolgot (és nem tévedtem).

17.3. Monológ. ***Oktató:*** Az SML könyvtárak használatáról

A könyvtári függvényeket általában a `Konyvtar.fuggveny` teljes – vagy másképpen *minősített*, angolul *qualified* – névvel lehet elérni, pl. `Int.fromString`. Fontos, hogy Unix/Linux alatt a kis- és nagybetűket a könyvtár- és függvénynevekben meg kell különböztetni, azaz `Int` és nem `int` vagy `INT`. A programok hordozhatósága érdekében az írásmódra Windows alatt is ügyelni kell. A pontos írásmódot a könyvtári függvényeket ismertető leírások tartalmazzák.

Ha a programjukat – pl. az mosmlc fordítóval – lefordítják le és összeszerkesztik (ezt teszi a kis- és nagyházik beadásakor a tesztkörnyezet is), akkor a könyvtárak használata általában nem jelent további technikai nehézséget.

Az mosml értelmező használatakor viszont a program betöltése előtt a felhasználandó könyvtárakat be kell tölteni, ezt az

```
app load ["Konyvtar1", "Konyvtar2", ...];
```

vagy egyetlen könyvtár betöltése esetén a

```
load "Konyvtar";
```

szerkezetű függvényalkalmazással tehetik meg. Példák:

```
app load ["Int", "Binaryset"];
```

```
load "Regex";
```

Fontos, hogy ezt a függvényhívást, ha a programot le fogják fordítani, ne írják bele a programjukba, mert az mosmlc fordító „elhasal” rajta.

Bizonyos gyakran használt könyvtárakat (pl. `String`, `List`) nem kell betölteni, mert induláskor az mosml értelmező ezeket automatikusan betölti.

Ha nem akarják kiírni egy-egy könyvtári függvény teljes nevét (ami egyébként hasznos, többek között javítja a program olvashatóságát), akkor az adott könyvtárat megnyithatják az `open Konyvtar` deklarációval, pl.

```
open Int;
```

Ettől kezdve a `min` hívás azonos lesz az `Int.min` hívással. Ezt a megoldást, mint írtuk, mégsem javasoljuk, mert nehezebben olvashatóvá teszi a programot.

17.4. Kérdés. Nem tudja valaki, hogy hogyan lehet egy SML könyvtárat betölteni (pl. `Real`), mert az ETS nem ismeri a mosml-ben használt `load`-ot?

17.4. Válasz. ***Oktató:*** Nem kell betölteni, elég a minősített nevet megadni (pl. `Math.sin`). A gyakran használt, azaz az `Int`, `Real`, `Char`, `String`, `List` stb. könyvtárak esetében a függvények döntő többsége elé a könyvtár nevét ki sem kell írni, mert ezeket belső függvényként kezeli az mosml.

A `load` csak az ún. interaktív módban, azaz mosml-ben használható (akárcsak a `use`). Az ETS az mosmlc fordítóval lefordítja a programot a futtatása előtt, és a szükséges modulokat automatikusan betölti.

17.5. Kérdés. Mi az oka annak, hogy bizonyos könyvtári listakezelő függvények, amelyek pl. a `list.sig`-ben le vannak írva, nem működnek, ha betöltöm a könyvtárat. Pl. `revAppend`, `mapPartial`, `last`, `nth`.

17.5. Válasz. ***Oktató:*** Alighanem az az oka, hogy a könyvtárnevet ki kell írni, mégpedig *nagy* kezdőbetűvel: `List.nth`; `List.revAppend`, mind Unix/Linux, mind Windows alatt. Más okát nem látom.

Megoldás lehet az is, hogy láthatóvá teszi a kérdéses könyvtár, pl. a `List` könyvtár tartalmát, így: `open List`.

17.6. Kérdés. Egy roppant egyszerű kérdésem van: hogy hívják azt a beépített függvényt, amely egy `int` argumentumra a hozzá tartozó `char` típusú értéket adja eredményül?

17.6. Válasz. ***Oktató:*** A függvény rövid neve `chr`, teljes neve `Char.chr`. Példa:

```
- chr(ord #"a" );
> val it = #"a" : char
```

17.7. Kérdés. Kérdésem az sml gyakorlórendszer egyik feladatához kapcsolódik: be kellene töltenem a `Real` könyvtárat (`load "Real"`), amit a gyakorlórendszer nekem nem szeret. Biztosan van valami más megoldás, amihez nem kell a `Real.fromInt`, de nem jövök rá.

17.7. Válasz. ***Oktató:*** A konverziót a `real`, teljes nevén a `General.real` függvénnyel is elvégezheti. De használhatja a `Real.fromInt` függényt is. A `Real` könyvtárat ehhez nem kell betöltenie, mert az mosmlc maga betölti, ha szüksége van rá. A `load` ugyanis, amint azt ebben a fejezetben másutt már leírtam, csak az interaktív mosml-környezetben használható.

17.8. Kérdés. Az lenne a nagy problémám, hogy írtam egy gagyi kis sml progit, de sajnos nem akar működni, és fogalmam sincs, hogy miért nem. Hibának kiírja hogy „kezeletlen kivétel: subscript” !!!???.

De vajon miért? Szerintem, amibe bele tud kötni, az ez a sor:

```
if String.sub(xs, String.size xs) = ...
```

Valószínűleg a `sub`-bal kötekszik, de miért? (Egyébként csak annyit még, hogy ez egy gyakorlórendszerbeli feladat, mégpedig a `komplista` nevű).

17.8. Válasz. ***Oktató:*** Természetesen azért kötekszik, mert érvénytelen indexet kapott. Ugyanis 0-tól kezdve indexel a `String.sub`, amint a következő részlet is mutatja:

```
- String.sub ("abc", 1);
> val it = #"b" : char
```

17.9. Kérdés. Most két függvényt nem értek és nem találok sehol: a `String.tokens`-t és az `as`-t. Mit csinálnak ezek? Hogyan kell őket használni?

17.9. Válasz. ***Oktató:*** A `String.tokens` és a hozzá hasonló `String.fields` függvény leírása megtalálható a `String` könyvtárban; ha telepítette az mosml-t, a saját gépén a `.../mosml/lib` mappában találja meg, egyébként a weben html-változatban is megtalálhatja a <http://www.dina.kvl.dk/~sestoft/mosmllib/String.html> címen.

Az `as` nem függvény, hanem kulcsszó. Lásd például az `(xxs as x::xs)` ún. *réteges mintában* (angolul: *layered pattern)*, ahol a teljes lista `xxs` néven, a feje `x`, a farka `xs` néven érhető el egy függvénydefinícióban annak a klóznak a törzsében, amelynek a fejében ez a minta szerepel.

17.10. Kérdés. Valaki el tudná magyarázni, hogyan is működik a `String.tokens` függvény?

17.10. Válasz. ***Hallgató:*** Valahogy így:

```
- String.tokens (fn #" " => true
                  | _ => false) "sfew rehwqo rejio341";
> val it = ["sfew", "rehwqo", "rejio341"] : string list
```

Tehát az első argumentum egy predikátum (azaz `bool` típusú eredményt adó függvény), amely megmondja, hogy mit tekintünk elválasztó karakternek – szeparátornak – (`true`), és mit nem (`false`); a második argumentum meg a tokenizálandó füzér (`string`). A függvény eredménye egy, a tokeneket tartalmazó, `string` típusú elemekből álló lista. Az egymás mellett álló szeparátorokat a függvény egyetlen szeparátornak tekinti; a szeparátorokat „megeszi”.

Remélem, segítettem. Egyébként érdemes próbálgatni mosml-ben, ha nem érted.

17.11. Kérdés. Nem tudja valaki, hogyan lehet két `int`-et binarisan AND-elni, illetve OR-olni?

17.11. Válasz. ***Oktató:*** `int` típusú (előjeles egész) értéket nem lehet, `word` és `word8` típusú (előjel nélküli egész) értéket lehet; lásd a `Word.andb`, `Word.orb`, `Word8.andb` és `Word8.orb` függvényeket. `fromInt` és `toInt` néven ugyanott konverziós függvények is vannak.

18. fejezet

Kiírás

18.1. Kérdés. Hogyan lehetne csak egyetlen klóz paramétereit kiíratni, amikor meghívódik? Hogyan lehet abortálásnál megtudni, hogy hol tartott a program, és mik voltak a változókban? Ezt pedig már kérdezték, csak elfelejtettem: hogyan lehet nagyobb mélységig kiíratni egy listát?

18.1. Válasz. ***Oktató:*** Tetszőleges típusú érték kiírására a `printVal` függvényt használhatja az SML-ben, de *csak interaktív módban.* Vagy használhatja a `print`-et, de az `string` típusú argumentumot vár. A `printDepth` frissíthető változót használhatja a kiírási mélység megváltoztatására.

A `print` a `TextIO.sig`-ben, a másik kettő a `Meta.sig`-ben van leírva. Az *MOSML Owner's Manual*-ben találja meg, hogyan állíthatja be `printDepth` értékét. Mindezek a letöltött mosml-csomagban megtalálhatók.

18.2. Kérdés. Meg tudná mondani valaki, hogy hogyan tudok nem toplevelen kiíratni valamit? Olyankor a `print`-re azt mondja, hogy *syntax error*.

18.2. Válasz. ***Oktató:*** A `print` (típusa: `string -> unit`) mindig működik. A `printVal` (típusa: `'a -> 'a`) az, ami *csak interaktív módban* működik, de az mosml *Syntax error*-t a `printVal`-ra sem mondhat. Ha mosmlc-vel megpróbálom lefordítani, ezt a hibaüzenetet kapom:

```
! val _ = printVal ["kukucs"];
!         ^^^^^^^^
! Unbound value identifier: printVal
! Uncaught exception:
! Fail  "compile: error(s) in the source program"
```

Valami mást ront el.

Figyelem: A `print` *mindig* `string` típusú paramétert vár!

19. fejezet

Hibakeresés, nyomkövetés

19.1. Kérdés. Sajnos nem emlékszem, hogy a Prolog *trace*"-eljárásához hasonló van-e az SML-ben. Jól jönne. De van egy olyan érzésem hogy nincs... Az mosml-ben hogyan kell debuggolni? Valahogyan beletracelni az SML-programba? Mert a nagyházi-ellenőrző nagyon informatív "Nulla megoldást találtam" eredményével nem sokat tudok kezdeni. :(

19.1. Válasz. ***Hallgató:*** Nincs, de jó helyen elhelyezett printekkel meg kivételkezeléssel nagyon frankón tudod követni a programjaid... :)

Oktató: Beépített hibakereső (*debugger*) vagy nyomkövető (*trace*) nincs az mosml-ben. :-(Csak a kőbaltás módszer van: `print` és `printVal` függvényhívásokat rakhat a programszövegbe, és ezekkel tetszőleges *debug*-üzeneteket írathat ki futás közben. Pl.

```
(print "f hívása\n"; printVal(f 1 #"2" "3"))
```

Ez nem túl szép (nem nevezhető deklaratívnak), de nagyon hasznos lehet. Ha a `print` helyett a `TextIO.output` függvényt használja, a nyomkövető-üzeneteket tetszőleges fájlba tudja írni.

Szerencsére az sml szigorú típusellenőrzése sok hibára fényt derít már fordítási időben. Sokat segít az is, ha a programját sok rövid, könnyen ellenőrizhető függvényből rakja össze: ha ezeket sikerül hibátlanul megírni, akkor a felhasználásukkal felépített függvények is rövidek, könnyen ellenőrizhetők maradnak, így az ezek felhasználásával felépített függvények is rövidek, könnyen ellenőrizhetők maradnak s.í.t.

Ha ragaszkodik a hibakeresőhöz és a nyomkövetőhöz, a Poly/ML-ben vannak ilyen segédeszközök (de nem olyan fejlettek, kényelmesek, mint a SICStus-ban). Lásd még:

- Összefoglaló a nyomkövetési lehetőségekről SML-ben:
  <http://dp.iit.bme.hu/sml/eloadas_anyagok/smlDebug-p1.pdf> (1 dia/lap),
  <http://dp.iit.bme.hu/sml/eloadas_anyagok/smlDebug-p2.pdf> (2 dia/lap),
  <http://dp.iit.bme.hu/sml/eloadas_anyagok/smlDebug-p4.pdf> (4 dia/lap)
- A Poly/ML-hez nyomkövetési példák:
  <http://dp.iit.bme.hu/sml/eloadas_anyagok/smlDebug.sml>
- Poly/ML debugging: <http://www.polyml.org/docs/Debugging.html>

19.2. Kérdés. Mit jelent az alábbi hibaüzenet? Váratlan fájlvége? Szerintem mindent lezártam, nincs befejezetlen ág, minden `if`-hez tartozik `then ... else ldots` Tanácstalan vagyok, nem tudom debuggolni:

```
File "K.sml", line 262, characters 0-1:
! <EOF>
! ^^^^^
! Syntax error.
```

19.2. Válasz. ***Hallgató:*** Valamit mégiscsak elfelejtettél lezárni. Azt javaslom, próbáld meg a fájlt darabokban odaadni az értelmezőnek. Legcélszerűbb felezéssel (először az első felét, ha nincs hiba, a háromnegyedét, ha van, a negyedét stb.)

A másik lehetőség, hogy beszúrsz pontosvesszőket a deklarációk végére, akkor jobban fogod látni, hogy mi az, ami még hibátlan, és mi az, amit már nem fogad el.

20. fejezet

SML-programok

20.1. Monológ. ***Oktató:*** Ígéretem szerint küldöm a legutóbbi konzultáción ismertetett feladatpárt (mu-lista generálása, ill. felismerése). A generálásra egy megoldást leírok, a felismerésre a megoldást meghagyom gyakorló feladatnak.

N-edrendű mu-listának hívjuk azt a listát, amelynek ilyen a szerkezete:

```
[m, u, m, u, u, m, u, u, u, ..., m, u, ..., u]
```

Azaz a lista pontosan `N` db `m` elemet tartalmaz, és az `i`-edik `m`-et pontosan `i` db `u` elem követi. Adott a `datatype mu = m | u` deklaráció.

(1) Írjon olyan SML-függvényt `mu` néven, amely N-edrendű mu-listát készít! Ügyeljen a hatékonyságra: ne használjon append-et!

```
(* mu N = a fenti definíció szerinti N-edrendű mu-lista
   mu : int -> mu list
   PRE: N >= 0
*)
```

Példa:

```
mu 5 = [m,u,m,u,u,m,u,u,u,m,u,u,u,u,m,u,u,u,u,u]
```

Egy lehetséges megoldás:

```
local
  (* gmu i j ls = i-szer egy-egy m és minden m után j db u;
                  j minden m után újraindul i aktuális értékétől
  *)
  fun gmu 0 _ ls = ls
  | gmu i 0 ls = gmu (i-1) (i-1) (m::ls)
  | gmu i j ls = gmu i (j-1) (u::ls)
in
   fun mu n = gmu n n []
end
```

(2) Írjon olyan SML-függvényt `mulista` néven, amely egy listáról eldönti, hogy mu-lista-e, s ha igen, hányadrendű! Ha a lista nem mu-lista, az eredmény -1 legyen. Ügyeljen a hatékonyságra!

```
(* mulista ls = ha ls mu-lista, akkor a rendszáma, egyébként ~1
  mulista : mu list -> int
*)
```

Példák:

```
mu [m,u,m,u,u,m,u,u,u,m,u,u,u,u,m,u,u,u,u,u] = 5
mu [m,u,m,u,u,m,u] = ~1
```

Feladat: írja meg a függvényt!

20.2. Kérdés. Az ETS-ben az *SML-programozás* szekcióban a platós (SPRG.19) feladatra a következő megoldást adtam:

```
fun platoseged (x1::x2::xs, (darab, elem)::ys) =
        if x1 = elem
        then platoseged(x2::xs, (darab+1, elem)::ys)
        else if x1 = x2
        then platoseged(xs, (2, x1)::(darab, elem)::ys)
        else platoseged(x2::xs, (darab, elem)::ys)
  | platoseged (x::xs, (darab, elem)::ys) =
        if x=elem
        then (darab+1, elem)::ys
        else (darab, elem)::ys
  | platoseged ([], ys) = ys
  | platoseged (_, _) = [];

fun plato (x::xs) = rev(platoseged(x::xs, [(0, x)]))
  | plato _ = [];
```

Lehet, hogy nem szép, de az mosml-be betöltve működik és (szerintem) helyes eredményt ad. Ennek ellenére a gyakorlórendszer szerint a függvény rossz eredményt ad.

Ezek után a kérdésem az lenne, hogy melyik az a teszteset, amelyikre ez nem működik (és amely miatt nem fogadja el tőlem a gyakorlórendszer).

És ha már itt tartunk, megjegyzem, hogy éppen az ilyen esetekben nagyon jól jönne, ha a gyakorlórendszer kiírná, hogy milyen bemenetre nem működött jól a program, mi volt a várt és a kapott eredmény. (A Prolog programozás szekcióban éppen így van, ha jól emlékszem.) Megvalósitható? Szerintem nagyon nagy segítség lenne…

20.2. Válasz. ***Oktató:*** !!! Sajnos, utólag nagyon nehéz megmondani, a tesztesetek már nincsenek meg, vagy ha meg is vannak, nem tudjuk, hol.

20.3. Kérdés. Az sml-gyakorlórendszer *plató* feladatára találtunk két megoldást. Az első inkább imperatív észjárással érthető meg, míg a másodiknál ragaszkodtunk ahhoz, hogy a vizsgához hasonlóan csak egy segédfüggvényt használjunk fel. Csak arra lennénk kíváncsiak, hogy mindkét megoldás teljes értékű-e!

Az első megoldás:

```
(* a lista elején álló azonos elemek számát adja vissza
*)
fun hanyszor (egy::stb, vmi) = if egy = vmi
                               then 1 + hanyszor(stb, vmi)
                               else 0
  | hanyszor ([egy], vmi) = 1
  | hanyszor ([], vmi) = 0;

(* levágja a lista elején álló azonos elemeket
*)
fun csonkol [] = []
  | csonkol [egy] = []
  | csonkol (egy::ketto::stb) = if egy = ketto
                                then csonkol(ketto::stb)
                                else ketto::stb;

(* maga a plató feladat
*)
fun plato (egy::ketto::stb) =
      if egy = ketto
      then (hanyszor(egy::ketto::stb,egy), egy) ::
                          plato(csonkol(egy::ketto::stb))
     else plato(ketto::stb)
|    plato [egy] = []
|    plato [] = [];
```

A második megoldás:

```
fun plato (x::y::xs) =
      let
          fun cutcount (x::y::xs, akku) =
                if x = y
                then cutcount(y::xs, x::akku)
                else if List.length(x::akku) = 1
                then cutcount(y::xs, [])
                else (List.length(x::akku), x) ::
                                        cutcount(y::xs, [])

            | cutcount ([x], akku)=
```

```
                if List.length(akku) = 0
                then []
                else [(List.length(x::akku), x)]
        in
           if x = y
           then cutcount(x::y::xs, [])
           else cutcount(x::y::xs, [])
        end
    | plato [x] = []
    | plato [] = []
```

20.3. Válasz. ***Oktató:*** !!! Nem látom, mi a különbség az `if`-ben a két változat között.

20.4. Kérdés. Írj az alábbi fejkommentet kielégítő SML-függvényt! Segítségül példát is mutatunk a használatára.

```
(* plato ls = az ls platóinak hosszából és a platókat
     alkotó elemekből álló párok listája. Platónak hívunk
     egy olyan, legalább kételemű részlistát, amely csupa
     azonos elemből áll, és egyik irányba sem terjeszthető
     ki ezzel az elemmel.
   plato : ''a list -> (int * ''a) list
*)
```

Példa:

```
plato(explode "abbbbaacbb") = [(4,#"b"), (2,#"a"), (2,#"b")]
```

20.4. Válasz. ***Hallgató:*** Az alábbi megoldás nem túl szép, de működik. :) Az `lst` függvény olyan listát csinál, amelyben minden karakterhez odaírja, hogy 1 van belőle, és párt képez belőle. Példa:

```
lst [#"a", #"a", #"b"] = [(1, #"a"), (1, #"a"), (1, #"a")]

fun lst [g] = [(1,g)]
  | lst (f::ff) = (1,f) :: lst ff
  | lst [] = []
```

Az `add` függvény összeadja az azonos csoportokat. Példa:

```
add [(1, #"a"), (1, #"a"), (1, #"a")] = [(2, #"a"), (1, #"a")]

fun add ((i, f) :: (1, ff) :: fff) =
      if f = ff
      then add((i+1, ff) :: fff)
      else (i, f) :: add ((1, ff) :: fff)
  | add [(i, g)]) = [(i, g)]
  | add [] = []
```

Az `irt` függvény kiirtja az `(1, valami)` tagokat a listából. `List.filter`-rel szebb lenne, de valahogy nem szereti a típusokat, én meg inkább megírtam.

```
fun  irt ((i, f) :: farok) = if i = 1
                             then irt farok
                             else (i, f) :: irt farok
  | irt [(i, f)] = if i = 1 then [] else [(i, f)]
  | irt [] = []

fun  plato ls = irt(add(lst ls))
```

Talán a következő egy picivel szebb megoldás… ;)

```
local
  fun platoi(A, N, X::XS) =
        if A = X
        then platoi(A, N+1, XS)
        else if N > 1
        then (N, A):: platoi(X, 1, XS)
        else platoi(X, 1, XS)
    | platoi (A, N, []) =
        if N > 1
        then [(N, A)]
        else []
   in
      fun plato (X::XS) = platoi(X, 1, XS)
        | plato [] = []
end
```

20.5. Kérdés. Én ezt nem értem. A feladat ez: Írj az alábbi fejkommentet kielégítő SML-függvényt! Segítségül példát is mutatunk az alkalmazására. (SPRG.16)

```
(* komplista : string -> string list
   komplista s = az s unix-állománynév komponenseinek a #"/"
     karakterek mentén felbontott listája. Az állománynévben
     egymás után többször szereplő #"/" jel egyetlen #"/"
     jelként veendő figyelembe. Az állománynév elején és/vagy
     végén álló #"/" jel esetén az eredménylista első és/vagy
     utolsó eleme az üres füzér ("") legyen.
*)
```

Példa:

```
komplista "dr-1/dr-2/file.ext" = ["dr-1","dr-2","file.ext"];
```

A megoldásom:

```
fun komplista s = String.fields (fn x => ord x=ord #"/") s
```

A mosml szerint jó (kipróbáltam a megadott példát is), a gyakrendszer szerint „lesz ez még jobb is”. Mit rontottam el???

Ui. Jó lenne, ha itt is kiírna valamit a program, mint a Prolognál, hogy mi volt hibás.

20.5. Válasz. ***Oktató:*** !!! Vajon a gyakorlórendszerben vagy a megoldásban van a hiba? Lehet hibát keresni, tippelni!

20.6. Monológ. ***Hallgató:*** Igen, mindenki jól látja, csak egy fájlt mellékeltem, mert sajnos ezzel szenvedtem az elmúlt egy órában, ha nem többen... de megszületett! Lehet, hogy nem a legszebb, lehet, hogy nem a legelegánsabb, de az enyém. :)

Írj az alábbi fejkommentet kielégítő SML-függvényt! Segítségül példát is mutatunk az alkalmazására.

```
(* komplista : string -> string list
   komplista s = az s unix-állománynév komponenseinek a #"/"
     karakterek mentén felbontott listája. Az állománynévben
     egymás után többször szereplő #"/" jel egyetlen #"/"
     jelként veendő figyelembe. Az állománynév elején és/vagy
     végén álló #"/" jel esetén az eredménylista első és/vagy
     utolsó eleme az üres füzér ("") legyen.
*)
```

Példa:

```
komplista "dr-1/dr-2/file.ext" = ["dr-1","dr-2","file.ext"];
```

Szenvedésem eredménye:

```
fun komplista s =
  if String.sub(s,0) = #"/" andalso
       String.sub(implode(rev(explode s)),0) = #"/"
  then "" :: String.tokens (fn c : char => ord c=47) s @ [""]
  else if String.sub(s,0) = #"/"
  then "" :: String.tokens (fn c : char => ord c =47) s
  else if String.sub(implode(rev(explode(s))),0) = #"/"
  then String.tokens (fn c : char => ord c = 47) s) @ [""]
  else String.tokens (fn c : char => ord c = 47) s;
```

20.7. Kérdés. Kérdésem az sml gyakorlórendszer 23-as feladatával kapcsolatos:

Írj az alábbi fejkommentet kielégítő SML-függvényt! Segítségül példát is mutatunk az alkalmazására.

```
(* harmKozep ms = az ms elemeinek harmonikus közepe, azaz az
     ms-beli elemek számának és reciprokösszegének hányadosa
   harmKozep : int list -> real
*)
```

Példa:

```
harmKozep [2, 4, 4] = 3.0
```

Kreáltam is megoldást rá:

```
fun harmKozep xs =
      let
          fun hk ([], k, p) = Real.fromInt k / p
            | hk (x::xs,k, p) =
                    hk (xs, k+1, p + (1.0 / Real.fromInt x ))
      in
         hk(xs, 0, 0.0)
      end
```

A Moscow ML-lel okosan működik, csakhogy ehhez be kell tölteni a `Real.uo`-t, de a `load "Real"`-t nekem a gyakorlórendszer nem szereti. Biztosan van valami más megoldás, amihez nem kell a `Real.fromInt`, de nem jövök rá, itt mindenképpen kell a `/` miatt. :-(Esetleg el lehet kerülni valahogy a `load`-ot?

20.7. Válasz. ***Hallgató:*** Használd helyette a `real` függvényt, ahhoz nem kell a `load`.

Oktató: A `load` használatáról már sokszor, sokat írtunk a GYIK más fejezeteiben. Nézze meg az *SML Basis Library*, *Az MOSML értelmező használata* és a *Kisbetű, nagybetű* című fejezeteket!

20.8. Kérdés.

```
(* osszeg(xs, ossz): az xs nemüres számlista elemei közé
     rakva az eredménylista elemeinek az elemeit ( op+ vagy
     op-) a kapott kifejezés értéke ossz; az eredménylista
     elemei eggyel rövidebbek xs-nél.
   osszeg : int list * int -> (int * int -> int) list list
*)
```

Példa:

```
osszeg([2,3,4,5,6,7], 13) =
          [[op+,op+,op+,op+,op-], [op-,op-,op+,op+,op+]
```

Tudja valaki ennek a megoldását?

20.8. Válasz. ***Oktató:*** Szabad a gazda!

20.9. Kérdés. Egy kérdésem lenne az `inter` függvénnyel kapcsolatban: a könyvben és a diákon található változat úgy működik, hogy ha az `xs` listában egy elem többször szerepel, azt az eredményben is többször fogja szerepeltetni. Ez jó így, vagy feltehetjük, hogy az `xs` és az `ys` is halmazok, tehát minden elem eleve csak egyszer szerepel bennük? Vagy inkább a sima felfűzés helyett a `newMem`-et kellene használni?

```
fun isMem (y, x::xs) = y = x orelse isMem(y, xs)
  | isMem (_, []) = false;
infix isMem;

fun newMem(x, xs) = if x isMem xs then xs else x::xs;

fun inter ([], _) = []
  | inter (x::xs, ys) = let val zs = inter(xs, ys)
                        in
                           if x isMem ys then x::zs else zs
                        end;
```

Az én változatom:

```
fun inter (x::xs, ys) = if x isMem ys
                        then newMem(x, inter(xs, ys))
                        else inter(xs, ys)
  | inter ([], _) = [];
```

20.9. Válasz. ***Oktató:*** Specifikáció kérdése. `isMem`-mel biztonságos, de drágább.

20.10. Monológ. ***Hallgató:*** Most csatolom az ETS-es SML-függvényeket, láttam, hogy érdekelne vkit. Az egyszerűbbekkel foglalkoztam most egyelőre, de meg kell hagyni, a nehezebbekkel nem is bírtam. ;) Szóval ha vkinek vannak jó megoldásai a nehéz példákra, akkor legyen szíves, küldje el vmelyik listára.

Ui. A Samirello-féle `komplista` megvan, az így átolvasva is elég nehezen emészthető, ja meg a `recOsszeg` az oké, azzal még elbírtam. ;)

ETS SML EGYSZERŰ FÜGGVÉNYEK by LiFeX (2004-01-27)

1. SML – egyszerű függvények írása – 8., közepes feladat (#2990)
   Írj az alábbi fejkommentet kielégítő SML-függvényt! Segítségül példát is mutatunk az alkalmazására.

   ```
   (* coded t = olyan füzér, amelyben minden t-beli karakter
        helyén az eggyel nagyobb ascii-kódú karakter van
        PRE: minden t-beli karakter kódja < 255
      coded : string -> string
   *)
   ```

Példa:

```
coded "van" = "wbo";
```

Megoldás:

```
fun coded t = let
                  fun b (x::xs) = chr ((ord x) +1)::b xs
                    | b _ = []
              in
                 implode(b (explode t))
              end;
```

2. SML – egyszerű függvények írása – 2., könnyű feladat (#2984)
   Írj az alábbi fejkommentet kielégítő SML-függvényt! Segítségül példát is mutatunk az alkalmazására.

```
(* len xs = az xs elemeinek a száma
   len : 'a list -> int
*)
```

Példa:

```
len ["a","ab","abc"] = 3;
```

Megoldás:

```
fun len [] = 0
  | len (x::xs) = 1 + len xs;
```

3. SML – egyszerű függvények írása – 4., könnyű feladat (#2986)
   Írj az alábbi fejkommentet kielégítő SML-függvényt! Segítségül példát is mutatunk az alkalmazására.

```
(* inc xs = olyan lista, amelyben minden xs-beli x elem
     helyén x+1 van
     PRE: xs minden eleme < Int.maxInt
   inc : int list -> int list
*)
```

Példa:

```
inc [4,3,8] = [5,4,9];
```

Megoldás:

```
fun inc [] = []
  | inc (x::xs) = (x+1)::inc xs;
```

4. SML – egyszerű függvények írása – 1., könnyű feladat (#2983)
   Írj az alábbi fejkommentet kielégítő SML-függvényt! Segítségül példát is mutatunk az alkalmazására.

```
(* ones ns = az 1 értékek száma ns-ben
   ones : int list -> int
*)
```

Példák:

```
ones [5,8,7] = 0;
ones [1,4,1,6,1,8,1] = 4;
```

Megoldás:

```
fun ones [] = 0
  | ones (x::xs) = if (x = 1)
                   then 1 + ones xs
                   else ones xs;
```

5. SML – egyszerű függvények írása – 12., közepes feladat (#2994)
   Írj az alábbi fejkommentet kielégítő SML-függvényt! Segítségül példát is mutatunk az alkalmazására.

```
(* sumli (ms, ns) = az ms és ns elemeinek páronkénti
     összeadásával előálló lista (a rövidebb listát
     képzeletben annyi 0-val egészítsd ki, hogy a két
     lista hossza azonos legyen)
   sumli : int list * int list -> int list
*)
```

Példa:

```
sumli([1,2,3], [4,5,6,9,7]) = [5,7,9,9,7];
```

Megoldás:

```
fun sumli ([],[]) = []
  | sumli ((x::xs),(y::ys)) = (x+y)::sumli(xs,ys)
  | sumli ((x::xs),[]) = x::sumli(xs,[])
  | sumli ([],(y::ys)) = y::sumli([],ys);
```

6. SML – egyszerű függvények írása – 3., könnyű feladat (#2985)
   Írj az alábbi fejkommentet kielégítő SML-függvényt! Segítségül példát is mutatunk az alkalmazására.

```
(* longer (xs, ys) = igaz, ha xs hosszabb ys-nál
   longer : 'a list * 'b list -> bool
*)
```

Példa:

```
longer([1,2,3], [true, false]) = true;
```

Megoldás:

```
fun longer ([],[]) = false
  | longer ((x::xs),[]) = true
  | longer ([],(y::ys)) = false
  | longer ((x::xs),(y::ys)) = longer (xs,ys);
```

7. SML – egyszerű függvények írása – 5., könnyű feladat (#2987)

   Írj az alábbi fejkommentet kielégítő SML-függvényt! Segítségül példát is mutatunk az alkalmazására.

   ```
   (* fib n = az n-edik Fibonacci-szám
      fib : int -> int
   *)
   ```

   Példa:

   ```
   fib 0 = 0;
   fib 1 = 1;
   fib 10 = 55;
   fib 20 = 6765;
   ```

   Megoldás:

   ```
   fun fib 0 = 0
     | fib 1 = 1
     | fib n = fib(n-1) + fib(n-2);
   ```

8. SML – egyszerű függvények írása – 15., nehéz feladat (#2997)

   Írj az alábbi fejkommentet kielégítő SML-függvényt! Segítségül példát is mutatunk az alkalmazására.

   ```
   (* recOsszeg n = az 1 és n közötti egészek reciprokának
                     az összege
        PRE: 0 < n
      recOsszeg : int -> real
   *)
   ```

   Példa:

   ```
   recOsszeg 3 = 1.83333333333
   ```

   Megoldás:

   ```
   fun recOsszeg 1 = 1.0
   | recOsszeg n = (1.0/real(n)) + recOsszeg(n-1)
   ```

9. SML – egyszerű függvények írása – 9., közepes feladat (#2991)

   Írj az alábbi fejkommentet kielégítő SML-függvényt! Segítségül példát is mutatunk az alkalmazására.

```
(* condensed t = a t karaktereit a formázó karakterek
     kivételével az eredeti sorrendben tartalmazó füzér
   condensed : string -> string
*)
```

Példa:

```
condensed " \t Moha \r\t\n med\r \n" = "Mohamed";
```

Megoldás:

```
fun condensed t =
      let
          fun con [] = []
            | con ls = List.filter Char.isAlpha ls
      in
          implode(con (explode t))
      end;
```

10. SML – egyszerű függvények írása – 11., közepes feladat (#2993)

    Írj az alábbi fejkommentet kielégítő SML-függvényt! Segítségül példát is mutatunk az alkalmazására.

```
(* harmKozep ms = az ms elemeinek harmonikus közepe, azaz
     az ms-beli elemek számának és reciprokösszegének a
     hányadosa
   harmKozep : int list -> real
*)
```

Példa:

```
harmKozep [2, 4, 4] = 3.0;
```

Megoldás:

```
fun harmKozep ms =
      let
          val sz = real(length ms)
          fun recs [] = 0.0
            | recs (x::xs) = (1.0/real(x)) + recs xs
      in
         sz/(recs ms)
      end;
```

11. SML – egyszerű függvények írása – 6., könnyű feladat (#2988)

    Írj az alábbi fejkommentet kielégítő SML-függvényt! Segítségül példát is mutatunk az alkalmazására.

```
(* fakt n = n!
     PRE: n >= 0
   fakt : int -> int
*)
```

Példa:

```
fakt 5 = 120;
```

Megoldás:

```
fun fakt 1 = 1
  | fakt n = n * fakt(n-1);
```

12. SML – egyszerű függvények írása – 7., könnyű feladat (#2989)
    Írj az alábbi fejkommentet kielégítő SML-függvényt! Segítségül példát is mutatunk az alkalmazására.

```
(* sumr rs = az rs elemeinek az összege
   sumr : real list -> real
*)
```

Példa:

```
sumr [1.2,3.5,10.0] = 14.7
```

Megoldás:

```
fun sumr [] = 0.0
  | sumr (n::ns) = n + sumr(ns)
```

21. fejezet

Fogások

21.1. Kérdés. Van egy SML-es kérdésem: egy paraméterként valamilyen struktúrát váró függvénynél hogyan lehet ennek a struktúrának az egyes elemeit elérni?

21.1. Válasz. ***Oktató:*** Nem világos, mit nevez struktúrának. Találgatok:

1. rekord (record; ezt nevezik a C-ben struktúrának),
2. ennes (tuple; ez az SML-rekord speciális esete),
3. struktúra (structure; ez modulfogalom, implementációs modult jelent).

Gondolom, nem az utóbbiról van szó, arról ugyanis a tárgyban keveset tanulunk, és a zárthelyin, a vizsgán, a feladatokban sem igazán kell használni (paraméterként biztosan nem).

Akkor nézzük az első két esetet. Egy függvényben a paramétereket a legkényelmesebb mintaillesztéssel elérhetővé tenni. Ha a paraméter összetett, akkor használhatunk

1. összetett mintát (általában ez az ajánlott, olvashatóbb megoldás),
2. (kiválasztó) függvényt.

Példák összetett mintára:

1. rekordminta

```
fun // {den = 0, ...}     = raise Domain
  | // {num = n, den = d} = (real n) / (real d);
```

   vagy

```
fun // {den = 0, ...} = raise Domain
  | // {num, den}     = (real num) / (real den);
```

2. ennesminta

```
fun iterlen ([], n) = n
  | iterlen (_::xs, n) = iterlen(xs, n+1);
```

Példák kiválasztófüggvény alkalmazására:

1. ha rekord a paraméter:

```
type rat = {num : int, den : int};
fun // (r : rat) = if #den r = 0
                   then raise Domain
                   else real(#num r)/real(#den r);
```

2. ha ennes a paraméter:

```
fun iterlen p = if null(#1 p)
                then #2 p
                else iterlen(tl(#1 p), #2 p + 1);
```

A kérdező folytatja: Konkrétan az is érdekel, hogy ha a struktúrának valamelyik eleme lista, az ilyen elemet hogyan lehet elérni. Ugyanúgy, mint általában, vagy másképp? Tud estetleg mutatni valaki ezekre egy-egy rövid programrészletet?

Oktató: Ugyanúgy, lásd a fenti példákat: mind az `x::xs` jellegű minta, mind a `hd`, `tl`, `null` stb. függvények használhatók. Általában javasolom a mintaillesztést. Természetesen lista lehet ennes, rekord vagy más lista eleme is.

21.2. Kérdés. Hogy lehet SML-ben egy számról eldönteni, hogy páros-e vagy páratlan? Van erre valami beépített függvény?

21.2. Válasz. ***Oktató:*** Beépített függvény nincs. De pl. könnyen megoldható a feladat az `fn x => x mod 2 = 0` lambda-függvénnyel.

21.3. Kérdés. Olyan problémába ütköztem, hogy van 2 függvényem, és egymást akarják meghívni, de az sml az elsőnél hibát ír jelez, mert ott még nincs deklarálva a második.

21.3. Válasz. ***Oktató:*** Kölcsönösen rekurzív függvényeket az `and` kulcsszóval elválasztva lehet definiálni. Példa:

```
fun even x = ...
and odd  x = ...
```

Lásd még a jegyzetben az *Egyidejű deklaráció* c. szakaszt, valamint a jelen GYIK más fejezeteit.

22. fejezet

Kivételek

22.1. Kérdés. Meg tudja mondani valaki, hogy mikor, ill. mitől áll elő a `Match` kivétel, és hogy hogyan tudom lekezelni?

22.1. Válasz. ***Hallgató:*** Akkor kapod ezt a kivételt, ha egy függvény fejében (illetve általában egy értékdeklarációban) nem kezeled le az összes elképzelhető esetet (erről kapod a *pattern matching not exhaustive* figyelmeztetést, nem véletlenül).

Lekezelni, ha valóban ezt akarod, természetesen egy `handle` kulcsszóval kezdődő kifejezéssel lehet. Én mégis a következőt javaslom (így a figyelmeztetéseket is elkerülöd, ami stilisztikailag is szebb):

1. deklarálj egy saját kivételt, például:

   ```
   exception MyError of string
   ```

2. minden olyan függvényben, amelyben szándékosan hagynál kezeletlen eseteket, vezess be egy utolsó, mindenre illeszkedő klózt (mindenesjellel az argumentumok helyén), amelyben jelezd ezt a kivételt, argumentumában a függvény nevével és lehetőleg a függény argumentumaival, pl.:

   ```
   fun f (x::xs) l = ...
     | f []      0 = ...
     | f xs      n = raise MyError ("f " ^ list2string xs ^
                                    " " ^ Int.toString n)
   ```

   Így, ha mégis ráfutna az utolsó klózra, a jelzett kivételből pontosan meg tudod állapítani, hogy melyik függvényben volt a hiba, és azt milyen argumentumokkal hívtad meg.

   Megjegyzés. A `list2string` függvény könyvtári függvényként nem létezik, neked kell megírnod, ha valóban ki akarod írni az argumentum értékét.

22.2. Kérdés. Hogyan lehet egyszerre több beépített kivételt is lekezelni, pl. a a`Subscript`-et, a `Match`-et és az `Empty`-t?

22.2. Válasz. ***Oktató:*** Nézze meg az előadásdiákon a kivételek kezeléséről szóló részeket! A `handle` kulcsszó után mindenféle minták megadhatók (akárcsak egy függvénydeklarációban vagy egy `case`-kifejezésben), a mindenesjel (_) is.

23. fejezet

Nagyzéhá, pótzéhá, pótpótzéhá

23.1. Monológ. ***Oktató:*** A tárgy funkcionális programozásról szóló részéből a nagyzéhá előtti SML-előadásokon elhangzottakat kell tudni a nagyzéhára, pótzéhára, pótpótzéhára egyaránt. Az előadásdiák letölthetők a tárgy honlapjáról.

A diákon szereplő belső és könyvtári függvények, köztük az egyszerű ki- és beviteli, valamint konverziós függvények ismerete mellett elvárjuk a `General`, `Int`, `Real`, `Math`, `Char`, `String` és `List` könyvtárbeli típusok, függvények és más értékek használatának ismeretét.

2004-től kezdve nem adunk fel típusegyenleteket, típuslevezetési feladatokat a zéhán. A négy SML feladat pontozása rendszerint 7-7-7-9, az első két feladat általában három-három részfeladatból áll, többnyire 2-2-3 a részpontozásuk.

Az első feladatcsoportban szintaktikailag helyes, de szemantikai szempontból hibás SML-kifejezésekről kell megmondani, hogy miért hibásak.

A második feladatcsoportban helyes SML-kifejezések értékét kell felírni a lehető legegyszerűbb alakban.

A harmadik feladatban néhány más függvényből (pl. lambda-jelöléssel megadott vagy magasabb rendű könyvtári függvényekből) felépülő SML-függvényt kell alkalmazni különféle értékekre, és meg kell adni az alkalmazás eredményét. Kérdés lehet, hogy mutassa be a függvény egyszerűsítési lépéseit mohó (applikatív sorrendű) vagy lusta (normál sorrendű) kiértékelés esetén.

Végül a negyedik feladatban SML-függvényt kell írni valamilyen lista bizonyos tulajdonságú elemeinek az összeadására, összeszorzására, kigyűjtésére stb.

Érdemes megnézni a korábbi feladatsorokat: a típusegyenletek kivételével a másik három fajta hasonló marad. Néhány példa, a zéhában hasonló jellegű, de ezeknél azért nehezebb feladatok várhatók.

1. fajta. Miért hibás (egyetlen hiba van a kifejezésben)?

```
(8, 3=4) = (ord #"x", Char.isSpace)
```

2. fajta. Mi `xs` értéke egyszerűsítés után?

```
val (x::xs) = hd [[2*5, 7 div 3], [round 8.0], []]
```

3. fajta. Adott az f függvény:

```
fun f g [] = [] | f g (x::xs) = map g x :: f g xs
```

Mi lesz az

```
f chr [[2*5, 7 div 3], [round 8.0], []]
```

függvényalkalmazás eredménye? Mutassa be az egyszerűsítési (behelyettesítési) lépéseket mohó kiértékelést feltételezve!

4. fajta. Növekvő futamnak nevezzük egy listában a lehető leghosszabb, azaz semelyik irányban nem kiterjeszthető, monoton növekvő részsorozatokat. Írjon `futamSzam` néven olyan SML függvényt, amely egy egészlista növekvő futamainak számát adja eredményül. Példa:

```
futamSzam [1,4,7,3,5,11,19,8,6,7,8] = 4
```

23.2. Kérdés. Tudna vki az előző feladatokra egy-egy példát írni? Előzo zéhákat nem is érdemes tanulmányozni SML-ből ezek szerint?

23.2. Válasz. ***Hallgató:*** Én csinálgattam valamit, remélem, még nem túl későn. Nem garantálom, hogy mind jó, de talán segít. Amit tudtam, ellenőriztem.

1. fajta. Miért hibás (egyetlen hiba van a kifejezésben)?

```
(8, 3 = 4) = (ord #"x", Char.isSpace)
```

A `3 = 4` típusa `bool`, a `Char.isSpace` típusa `char -> bool`: különböző típusú értékek nem hasoníthatók össze, egyenlőségük nem vizsgálható.

2. fajta. Mi az `xs` értéke egyszerűsítés után:

```
val (x::xs) = hd [[2*5, 7 div 3], [round 8.0], []]
```

Kezdjük a jobb oldal egyszerűsítésével:

```
hd [[10, 2], [8], []]
```

tehát `(x::xs) = [10, 2]`, ugyanis a `hd` függvény a lista fejét veszi, amely itt `[10, 2]` tehát `xs = [2]`.

3. fajta. Adott az `f` függvény:

```
fun f g [] = [] | f g (x::xs) = map g x :: f g xs
```

Mi lesz az

```
f chr [[2*5, 7 div 3], [round 8.0], []]
```

függvényalkalmazás eredménye? Mutassa be az egyszerűsítési (behelyettesítési) lépéseket mohó kiértékelést feltételezve!

A `chr` függvénynek egy szám az argumentuma, az eredménye pedig az a karakter, amelynek az adott szám az ascii-kódja. Pl. `chr 120 = #"x"`. Az `f` függvény üres listára üres listát ad eredményül. Ha a lista nem üres, akkor az `x` lista összes elemére kiértékeli a `g` függvényt (ami itt nem más, mint a `chr` függvény), majd ezt az így keletkezett listát egy új lista elejére fűzi, és végül a maradék listára is meghívja ugyanezt.

Tehát a mohó kiértékelés miatt először beír mindent, amit lehet:

```
f chr [[10, 2], [8], []]
```

Utána először végigmegy a `[10, 2]` listán, és az elemeit átalakítja karakterekké. Az ascii-kódtáblából kiolvasható, hogy `chr 10 = #"\n"`, `chr 2 = #"\^B"`, `chr 8 = #"\b"`. Tehát kapok egy olyan listát, hogy `[#"\n", #"\^B"]`. Ezután jön a [8] lista, ebből `[#"\b"]` lesz. Végül `[]`-ből `[]` lesz. Tehát egy ilyen listát kapok: `[[#"\n", #"\^B"], [#"\b"], []]`.

4. fajta. Növekvő futamnak nevezzük egy listában a lehető leghosszabb, azaz semelyik irányban nem kiterjeszthető, monoton növekvő részsorozatokat. Írjon `futamSzam` néven olyan SML függvényt, amely egészlista növekvő futamainak számát adja eredményül.

Példa: `futamSzam [1,4,7,3,5,11,19,8,6,7,8] = 4`

A megoldásom ***majdnem*** jó! Egy hiba van vele: a `futamSzam [0]` meghívásra 2-t ad ki 1 helyett.

```
fun futamSzam [] = 0
  | futamSzam (x::xs) =
      let
         fun futam ([], Elozo, X) = X+1
           | futam ((x::xs), Elozo, X) =
                if x > Elozo
                then futam(xs, x, X)
                else futam(xs, x, X+1)
      in
         futam((x::xs), 0, 0)
      end
```

A kérdező folytatja: Ez a hiba könnyen kijavítható: `| futamSzam [n] = 1` berakásával, viszont pl. `futamSzam [1,1,1]`-re `3`-at ad, ami annyira nem jó…

Ki tudná kijavítani???

24. fejezet

Nagyházi, kisházi

24.1. Kérdés. Nem boldogulok a keretprogram futtatásával. A `KCsiga.sml` keresi a `unit Csiga`-t. Nekem van egy `Csiga.sml` fájlom. Mit tegyek???

24.1. Válasz. ***Oktató:*** Fordítsa le az mosmlc fordítóval az összes modult, valahogy így:

```
mosmlc -c TCsiga.sml Csiga.sig KCsiga.sig KCsiga.sml Csiga.sml
```

vagy egyesével is lehet (a sorrend nem közömbös! azt hiszem, ez a jó):

```
mosmlc -c TCsiga.sml
mosmlc -c Csiga.sig
mosmlc -c KCsiga.sig
mosmlc -c KCsiga.sml
mosmlc -c Csiga.sml
```

A -c kapcsoló hatására csak fordít, nem szerkeszt (vö. gcc).

Interaktív módban, azaz az mosml-lel a `Meta.compile : string -> unit` függvény alkalmazásával is le lehet fordítani a programot,.

A fordítás után (kizárólag az *interaktív módban!)* az

```
app load ["TCsiga", "Csiga", "KCsiga"];
```

betölti a teljes programot az mosml-be. A `KCsiga`-modulban lévő `f` függvényt `KCsiga.f` néven érheti el. Vagy az

```
open KCsiga
```

deklaráció érvényességi körében a rövid `f` név használata is elég. (Az `f` helyett természetesen létező függvények nevét kell használnia.)

24.2. Kérdés. Úgy érzem, a célegyenesben vagyok az SML téren is, de felmerült néhány kérdés bennem.

Első kérdésem: hogyan lehet SML esetén a programból gondoskodni a `list` könyvtár betöltéséről és megnyitásáról?

24.2. Válasz. ***Oktató:*** *Interaktív esetben,* otthoni élesztéskor, teszteléskor (mosml): `app load ["List"]` (a listában több könyvtár neve is megadható), vagy egyszerűen: `load "List"`. Vigyázat: a kis- és nagybetűket meg kell különböztetni!

Nem interaktív esetben, pl. a beadandó program esetén (mosmlc): semmit nem kell írni, tenni, a „főmodullal” együtt az összes szükséges modult betölti a futtatórendszer. Magyarán: a fogadószkript lefordítja a kapott `Lepeget.sml` fájlt (a specifikációjának ki kell elégíteni a `Lepeget.sig` szignatúrát, amelyet azonban nem kell beküldnie!), betölti (vele együtt betöltődik minden, ami kell), és futtatja.

Ha valaki több fájlra bontja a megoldását, akkor minden állományt be kell küldenie (.sml és esetleg .sig kiterjesztéssel), ezeket a fogadószkript mind lefordítja és betölti.

A kérdező folytatja: Második kérdésem: mennyire kell szoros időkorlátokra számítani? Azért kérdezem, mert az SML programom 12 hosszúságú bemenetekre már kb. 20 másodpercet molyol, amit elég soknak érzek a Prolog programom 2-3 másodpercéhez képest.

Oktató: Valóban indokolatlanul sok a Prologhoz képest. Az időkorlátba várhatóan mégis bele fog férni, 12 elemű sorozatra most 60 s-ot adunk. De hétfő estig még javíthat a programján…

A kérdező folytatja: Harmadik kérdésem: mit jelent az, hogy használjunk értelmes neveket? Nekem ez néha nagyon nehéz feladatnak tűnt… Az elég, ha számomra értelmesek a nevek? ;-)) Nyilván nem, de ezt a kitételt nagyon szubjektívnek érzem.

Oktató: Valóban nehéz, ezért gyakoroltatjuk. ;-) Másszóval azt várjuk el, hogy kifejező neveket használjanak. A programok általában (nem a hf-ekre gondolok) hosszú életűek, rendszerint túlélik a programozójukat a cégnél. Olyan neveket kell használni, amelyeket majd a fejlesztő utódok is viszonylag könnyen megértenek. Ezért kell persze a megfelelően áttekinthető, a program felépítését, szerkezetét magyarázó dokumentáció is.

A kérdező folytatja: Negyedik kérdésem: az SML-ben hogyan lehet rendezni egy, a nagyháziban használt bemenő listát? Tudom, hogy a függvény bemenete eleve rendezett, csak kíváncsi vagyok…

Oktató: Nézze meg a `Listsort.sig`-ben a `Listsort.sort` használatát. Vagy nézze meg a jegyzetben a listák rendezéséről szóló fejezetet, és írja meg a rendezőprogramot.

24.3. Kérdés. Elég leadni egyetlen `.pl` és egyetlen `.sml` fájlt, ha azokban minden jól megvan? SML esetén kell-e az `.sml`-hez mindenképpen `.sig`-et mellékelni? Vagy elég lesz, ha csak egy `.sml`-t küldök be, szépen kommentezve, benne pár függvénnyel?

24.3. Válasz. ***Oktató:*** Igen. A `.sig` tényleg nem kell, de kell a kétféle nyelven készült programokhóz a közös dokumentáció a szépen és érthetően fejkommentezett SML-függvények és Prolog-eljárások mellett.

24.4. Kérdés. Az SML házim a visszaigazolás alapján nem fordul le. Íme a hibaüzenet:

```
Testing SML programs
~~~~~~~~~~~~~~~~~~~
Moscow ML version 2.00 (June 2000)
  compiling and linking Lepeget ...
Makefile:31: .depend: No such file or directory
/opt/mosml20/bin/camlrunm /opt/mosml20/tools/mosmldep > .depend
/opt/mosml20/bin/mosmlc -c Lepeget.sml
File "Lepeget.sml", line 64, characters 16-25:
! |mgdbolo(mo,n)=(list.take(mo,n))::mgdbolo((list.drop(mo,n)),n)
!                 ^^^^^^^^^
! Cannot find file list.ui
make: *** [Lepeget.uo] Error 2
!ERROR: compilation and linking of Lepeget failed.
```

Mit kéne még elküldenem, hogy leforduljon? (A `list.ui`, `list.uo`, `list.sig` stb. fájlok elküldésével már próbálkoztam.)

24.4. Válasz. ***Hallgató:*** A hibaüzenet azt mondja, nem találja a `list.ui` fájlt. Azért nem találja, mert *nincs!* Van viszont `List.ui`. Tehát semmi mást nem kell beküldened, csak pontosan úgy kell írnod az alapkönyvtárak nevét, ahogy a dokumentációban szerepelnek. (A UNIX rendszerek figyelembe veszik a kis- és nagybetűk közti különbséget.)

24.5. Kérdés. Az a kérdésem, hogy elfogadható-e, ha az sml értelmező egy csomó *warning*-ot ad a házi feladatomra, vagy *warning*-mentesnek kell lennie?

24.5. Válasz. ***Oktató:*** Elfogadható, de növekszik a veszélye, hogy valamit elnéz, amit érdemes lenne meggondolnia.

24.6. Kérdés. Nem tudom lefordítani az alábbi egyszerű SML progit:

```
fun nyilak _=
      [([se,s,se,sw],[w,nw,nw,w],[ne,n,ne,nw],[se,ne,se,e]),
       ([se,s,se,sw],[w,nw,nw,w],[nw,n,nw,nw],[se,ne,se,e])];
```

ezzel a sorral:

```
mosmlc -c TNyil.sml Nyil.sig Nyil.sml KNyil.sig KNyil.sml
```

mert ezt kapom:

```
File "nyilak.sml", line 3, characters 16-18:
! fun nyilak _=[([se,s,se,sw],[w,nw,nw,w],[ne,n,ne,nw],...
!                 ^^
! Unbound value identifier: se
```

24.6. Válasz. ***Oktató:*** Nem a parancssorral van gond. Az `se`, `s`, `sw` stb. *adatkonstruktorállandók* a `TNyil` modulban vannak definiálva, alapesetben csak a *teljes nevükkel* érhetők el, azaz `TNyil.se`, `TNyil.s`, `TNyil.sw` stb. néven. Hogy ne kelljen annyit írni, a modul *megnyitható* az `open TNyil` deklarációval, utána már használhatja a rövid neveket is.

Megjegyzés. Az `open` éppen olyan deklaráció, mint bármelyik más abból a szempontból, hogy `local ... in ... end` deklarációk vagy `let ... in ... end` kifejezések deklarációs részében is használható, és ilyenkor a hatása természetesen lokális. Például

```
local open TNyil
in
      fun fuggv x y z = ...
end
```

Ui. A beadáskor figyeljenek arra, hogy a `.sml` fájl és a könyvtárak nevét nagy kezdőbetűvel írják, ugyanis a Un*x rendszerek megkülönböztetik a kis- és nagybetűket!

24.7. Kérdés. Az SML házi fordítása közben a következő üzenetet kaptam:

```
File "Kigyo.sml", line 92, characters 8-12:
!    val [A,B] = if length(L)=2 then L else [(0,0),(0,0)]
!         ^^^^
! Warning: pattern matching is not exhaustive
```

Azt értem, hogy mivel nem biztos, hogy az `L` pontosan 2 elemű, ezért *warning*-ot ad (gondolom, hogy az éppen erre vonatkozó feltételvizsgálatot nem tudja figyelembe venni). Hiba a feltétel miatt nem lép fel, csak ez a *warning* zavar. Hogyan lehet egyszerűen megszüntetni, illetve gond-e ez a *warning* a házi elfogadhatósága szempontjából? (Mivel a fogadórendszer elfogadja a programot, csak jelzi a *warning*-ot, úgy gondolom nem probléma, csak csúnya.)

24.7. Válasz. ***Oktató:*** A figyelmeztetések *(warning*-ok) nem befolyásolják a házi feladatra adott pontszámot.

Sajnos, listák esetén az ilyenfajta mintaillesztési *(pattern match)* figyelmeztetéseket csak magának a mintaillesztésnek a mellőzésével lehet elkerülni.

Tegyük fel, hogy az alábbi függvény fordításakor kapta a figyelmeztetést:

```
fun f L = let val [A, B] = if length L = 2
                           then L
                           else [(0,0),(0,0)]
          in
             (A, B)
          end;
```

Az alábbi változat nem szebb, de legalább figyelmeztetés nélkül lefordul:

```
fun f L = let val (A, B) = if length L = 2
                           then (hd L, hd(tl L))
                           else ((0,0),(0,0))
          in
             (A, B)
          end;
```

25. fejezet

Prologból SML?

25.1. Kérdés. Mostanáig Prologgal küzdöttem, most pedig nekiláttam az SML-nek. Egyelőre nem sok sikerrel. Azt hiszem, ott rontom el, hogy nem tudok elszakadni a prologos megközelítéstől. Prologban a `lepeget(...)` eljárás megírásakor a `member(...)` eljárást felhasználhattam arra, hogy egy ténylegesen lépegető algoritmussal az összes lehetséges pontba ellépjek. Ennek a megoldásnak mind külön ágai voltak. SML-ben ugyan alkalmazhatok egy függvényt egy lista minden elemére (pl. `map`-et), de annak az eredményei egy listában jönnek létre, nem pedig mint önálló elágazások.

25.1. Válasz. ***Oktató:*** Ajánlom figyelmébe a Prolog-jegyzet 4.5. alfejezetét *(Megoldások gyűjtése és felsorolása)*. Ott láthat egy semát arra, hogyan lehet felsoroló eljárást gyűjtőve átalakítani a Prologban. Persze ehhez tényleg át kell alakítania a programja szerkezetét, tehát nem használhatja a `member`-t és más beépített eljárásokat a felsorolásra. Ezek helyett egy

```
kovetkezo(+V0, +Param, -E, -V)
```

eljárást kell írnia, amelynek a bemenő paraméterei a kövekezők:

- `V0` a keresés állapotát leíró Prolog kifejezés,
- `Param` valamilyen, a keresés során nem változó paraméter;

a kimenők pedig:

- `E` a következő megoldás,
- `V` a keresés új állapota.

Ha egy ilyen segédeljárásra építve oldja meg a feladatot Prologban, akkor ezt a megoldást könnyen át tudja ültetni SML-be is.

25.2. Kérdés. Meg tudna mondani valaki, hogy az SML-ben hogyan lehet lekérdezni egy értékről, hogy változó-e? A prologos `var(X)` megfelelőjére gondolok.

25.2. Válasz. ***Oktató:*** Az SML-ben a változónak *mindig van* értéke, tehát a Prolog `var(X)` vizsgálat megfelelője mindig meghiúsulna, ha lenne: ezért nincs is ilyen vizsgálat).

A *hagyományos* funkcionális nyelvek, mint pl. az SML, nem ismerik a *logikai változó* fogalmát, de az újabb funkcionális nyelvekben (pl. *Oz, Alice)* megjelenik ez a nyelvi elem is.

26. fejezet

Vizsga

26.1. Kérdés. Valaki legyen olyan kedves és ossza meg velem, hogy nagyjából milyen kérdések várhatók SML-ből „nagyfeladat” címén! Nem baj, ha van hozzá valami megoldási útmutató is . . . a honlapon ugyanis csak Prolog vizsgafeladatokat találtam. :(

26.1. Válasz. ***Oktató:*** Vizsgapéldákat minden évben több előadáson is megbeszéltünk. Sok megjegyzésekkel fűszerezett SML-program van a <http://dp.iit.bme.hu/dp00s/sml-ea> címen a honlapon 2001-ből. Sok vizsgajellegű példa van a jegyzet 4. kiadásában is függelékben.

26.2. Kérdés. Az ETS-ben sok olyan feladat van, hogy valamilyen típusegyenletet kell megoldani. Ilyesmiről az előadásokon ritkán volt szó, a diákon legalábbis alig van nyoma. A jegyzetből sem sok derül ki a számomra. Akkor most kell ez a zéhára és utána a vizsgára?

26.2. Válasz. ***Oktató:*** A korábbi évekhez képest 2003. óta kevesebbet foglalkozunk típusegyenletekkel. A zéhán és a vizsgán is csak egyszerű kérdések lesznek a ebből a témaköréből. Természetesen a típusok, típuskifejezések, típusegyenletek fogalmát ismerni kell, hiszen kellő mélységű ismeretük nélkül az erősen típusos SML nyelven programot írni sem lehetne.

26.3. Kérdés. Tegnap a vizsgán elfelejtettem megkérdezni, hogy mi is volt a második hiba az alábbi sml feladatban. Az `ord "a"` volt az egyik, mert az `ord` típusa `char -> int`, és az `"a"` string típusú. De mi volt a másik?

```
(ord #"a", 2=3, Math.sin) = (ord "a", true, Math.cos)
```

26.3. Válasz. ***Hallgató:*** `Math.sin = Math.cos` a hiba, mert függvények egyenlősége nem vizsgálható (függvényérték típusa soha nem egyelőségi típus).