

Az SML egy résznyelvének szintaxisa és szemantikája

A 2005/06-os tanév őszi félévében a funkcionális programozásról szóló két első előadáson az SML nyelv alábbi elemeit ismertük meg.¹

Szavak

1. Állandók, pl. 3, 56, 1.7, 1.7878E45, false, true, ().
2. Operátorok, pl. +, -, *, /, div, mod, quot, rem, <, <=, >, >=, =, <>, ~.
3. Kulcsszók, pl. val, fun, if, then, else, let, in, end, int, bool, unit, =, (,), :, ->, *, #, (*, *), ,, i.
4. Azonosítók (nevek)
 - (a) Egyszerű azonosítók, pl. x, power.
 - (b) Összetett azonosítók, pl. Math.pi, Math.sqrt.

¹G. Smolka előadásjegyzete alapján (lásd <<http://www.ps.uni-sb.de/courses/prog-ws04/script>>) összeállította Hanák Péter. 2005. okt. 9.

Mondatok

1. Kifejezések
 - (a) Elemi kifejezések: egyetlen állandóból vagy azonosítóból állnak.
 - (b) Alkalmazások
 - i. Operátoralkalmazások: <monadikus operátor> <kifejezés>
 - ii. Operátoralkalmazások: <kifejezés> <diadikus operátor> <kifejezés>
 - iii. Projekciók: # <pozitív egészállandó> <kifejezés>
 - iv. Eljárásalkalmazások: <kifejezés> <kifejezés>
 - (c) Feltételes kifejezések (feltételből, következményből és alternatívából álló hármasok): if <kifejezés> then <kifejezés> else <kifejezés>
 - (d) Ennesek (legalább két tagúak): (<kifejezés> , ... , <kifejezés>)
 - (e) Kifejezések lokális deklarációval: let <program> in <kifejezés> end
2. Deklarációk (fejből és törzsből állnak)
 - (a) val <minta> = <kifejezés>

(b) fun <azonosító> <minta> = <kifejezés>

(c) fun <azonosító> <minta> : <típus> = <kifejezés>

3. Minták

(a) Értékminták

i. <azonosító>

ii. (<azonosító> , ... , <azonosító>)

(b) Argumentumminták

i. (<azonosító> : <típus> , ... , <azonosító> : <típus>)

4. Típusok

(a) Atomi típusok: int, real, bool, string, unit

(b) Eljárástípusok: <típus> -> <típus>

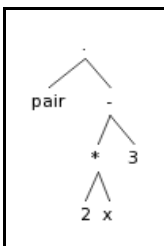
(c) Ennestípusok: <típus> * ... * <típus>

5. Programok: deklarációsorozatok (beleértve az üres deklarációsorozatot).

Mondatok két- és egydimenziós ábrázolása

A pair eljárás egy alkalmazásának ábrázolása

két dimenzióban, grafikusán:



egy dimenzióban, karakterekkel:

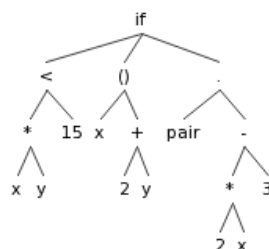
pair(2*x-3)

szavakkal (lexikális egységekkel):

pair (2 * x - 3)

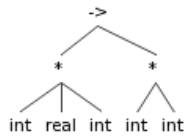
if x*y<15then(x,2+y)else pair(2*x-3)

if x * y < 15 then (x , 2 + y) else pair (2 * x - 3)



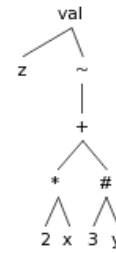
Egy kifejezés ábrázolása karakterekkel, szavakkal és fával

int * real * int -> int * int



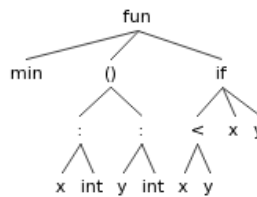
Egy típuskifejezés ábrázolása szavakkal és fával

val z = ~(2 * x + # 3 y)



Egy értékdeklaráció ábrázolása szavakkal és fával

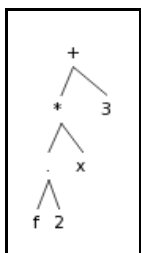
fun min (x : int , y : int) = if x < y then x else y



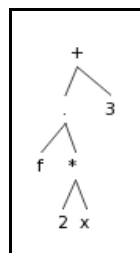
Egy eljárásdeklaráció ábrázolása szavakkal és fával

Zárójelek használata

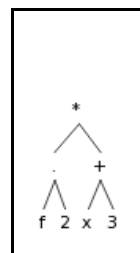
A kétdimenziós ábrázolás egyértelmű, az egydimenziós zárójelezéssel tehető azzá. Pl.



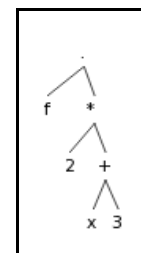
f 2 * x + 3



f(2*x) + 3



(f 2) * (x+3)



f(2*(x+3))

Szabad felesleges zárójeleket kitenni:

2*f~2+3 2*(f)~2+3 2*(f~2)+3 (2*(f~2))+3 ((2*(f~2))+3)

Kifejezések legfontosabb zárójelezési szabályai, példákön:

3*4+5 ≡ (3*4)+5 A multiplikatív műveletek megelőzik az additívakat

3+4+5 ≡ (3+4)+5 Az aritmetikai operátorok jobbra kötnek

3-4+5 ≡ (3-4)+5

f 3+4 ≡ (f 3)+4 Az eljárásalkalmazások megelőzik az operátorokét

f g 3 ≡ (f g)3 Az eljárásalkalmazások jobbra kötnek

Típuskifejezések legfontosabb zárójelezési szabályai, példákkal:

$\text{int} * \text{int} \rightarrow \text{int}$	\equiv	$(\text{int} * \text{int}) \rightarrow \text{int}$	A keresztszorzásnak van elsőbbsége.
$\text{int} \rightarrow \text{int} \rightarrow \text{int}$	\equiv	$\text{int} \rightarrow (\text{int} \rightarrow \text{int})$	A leképzés jobbra köt.
$\text{int} * \text{int} * \text{int}$	\neq	$(\text{int} * \text{int}) * \text{int}$	A keresztszorzás semerre nem köt!
$\text{int} * \text{int} * \text{int}$	\neq	$\text{int} * (\text{int} * \text{int})$	

Szabad azonosítók és környezetek

Az $x+y$ kifejezésben x is, y is *szabad azonosító*. $x+y$ csak akkor értelmezhető, azaz $x+y$ -nek csak akkor van értéke, ha x -nek is, y -nak is van értéke az adott környezetben. A *környezet* az azonosítókötések összessége.

$x+y$ pl. értelmezhető az $[x := 5, y := 7]$ környezetben, ahol $x = 5$ és $y = 7$.

A $\text{fun } q(x:\text{int}) = 3+(p\ x)$ kifejezésben a p szabad, az x pedig *kötött* azonosító (a matematikai logika terminológiájával: az x kvantifikált változó a q kvantorhoz van kötve). A deklaráció akkor hajtható végre, ha p -nek van értéke az adott környezetben.

A szabad változót tartalmazó mondatot *nyitottnak*, a szabad változót nem tartalmazót *zárt*nak nevezzük.

7

Most néhány további példát mutatunk be. Nézzük a következő programot:

```
fun p (x:int) = x
fun q (x:int) = 3+(p x)
```

A p deklarációja és a teljes program zárt, a q deklarációja önmagában azonban nyitott lenne.

A $\text{val } x = x$ deklaráció *nyitott*, mert az ilyen deklaráció nem rekurzív, azaz a törzsben szereplő x nem azonos a deklarálandó x -szel.

A $\text{fun } f(x:\text{int}):\text{int} = \text{if } x < 1 \text{ then } 1 \text{ else } x * f(x-1)$ deklaráció ezzel szemben *zárt*, ui. a fun -deklaráció rekurzív, és így a törzsben szereplő f azonos a deklarálandó f -fel.

Eljárások ábrázolása hármass formájában

A $\text{fun } p(x:\text{int}) = x+a$ deklaráció nyitott, önmagában nem deklarál semmit. Egy eljárásdeklaráció végrehajtásához az eljárás kódja mellett szükség van arra a környezetre is, amely a szabad változóihoz értéket köt, és ismerni kell az eljárás típusát is.

Egy eljárást tehát a $P = (C, T, E)$ egyenlettel írhatunk le, ahol a (C, T, E) hármass elemei rendre a kódot, a típust és a környezetet jelentik.

8

Nézzük például a

```
val a = 2*7
fun p (x:int) = x + a
```

programot. A deklarációk végrehajtása a p azonosítót a

```
(fun p x = x + a, int -> int, [a := 14])
```

hármashoz köti. A

```
val a = 2*7
fun p (x:int) = x + a
fun q (x:int) = x + p x
```

program végrehajtása az alábbi kötéseket hozza létre, ahol q is zárt, mert tartalmazza p -t:

```
a := 14
p := (fun p x = x + a, int -> int, [a := 14])
q := (fun q x = x + p x, int -> int,
      [p := (fun p x = x + a, int -> int, [a := 14])])
```

9

Nézzük a következő programot és a végrehajtásával létrehozott kötéseket:

```
fun p (x:int) = x
fun q (x:int) = p x
fun p (x:int) = 2 * x
val a = (p 5, q 5)

p := (fun p x = 2 * x, int -> int, [])
q := (fun q x = p x, int -> int,
      [p := (fun p x = x, int -> int, [])])
a := (10, 5)
```

mert q -t még p első kötésével hozta létre a program.

Egy eljárás törzse tehát azokat a kötéseket őrzi meg, amelyek az eljárás deklarálásakor voltak érvényben. Ezt a módszert *statikus* vagy *lexikális kötésnek* nevezik.

Szemantikai helyesség

Az értelmezőprogram a végrehajtása előtt ellenőrzi egy program szemantikai helyességét. Egy szemantikusan helyes programnak mindenképp előtte *zárt*nak és *típushelyesnek* kell lennie.

Típuszabályok

Diadikus aritmetikai operátorok:	$\frac{e_1 : t_1 \quad \circ : t_1 * t_2 \rightarrow t_3 \quad e_2 : t_2}{e_1 \circ e_2 : t}$
Feltételes kifejezések:	$\frac{e_1 : \text{bool} \quad e_2 : t \quad e_3 : t}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 : t}$
Enneskifejezések:	$\frac{e_1 : t_1 \quad \cdots \quad e_n : t_n}{(e_1, \dots, e_n) : t_1 * \cdots * t_n}$
Eljárásalkalmazások:	$\frac{e_1 : t_1 \rightarrow t_2 \quad e_2 : t_1}{e_1 e_2 : t_2}$

Végrehajtás

Most a megismert mondatfajták végrehajtását tárgyaljuk. A megoldandó feladatok:

Kifejezések végrehajtása. Adva van az e kifejezés és a V környezet, meg kell határozni az e értékét a V környezetben.

Eljáráshívások végrehajtása. Adva van a p eljárás és a v érték, meg kell határozni a p e értéket.

Deklarációk végrehajtása. Adva van a D deklaráció és a V környezet, meg kell határozni azt a környezetet, amelyet a D hoz létre a V környezetben.

Programok végrehajtása. Adva van a P program és a V környezet, meg kell határozni azt a környezetet, amelyet a P hoz létre a V környezetben.

Adjunkciónak nevezzük azt a $V_1 + V_2$ műveletet, amely V_1 -ből és V_2 -ből előállítja a V környezetet, mégpedig úgy, hogy

1. ha egy x azonosító csak V_1 -ben fordul elő, akkor az x a V_1 -beli értékhez lesz kötve V -ben is,
2. ha egy x azonosító csak V_2 -ben fordul elő, akkor az x a V_2 -beli értékhez lesz kötve V -ben is,
3. ha egy x azonosító mind V_1 -ben, mind V_2 -ben előfordul, akkor az x a V_2 -beli értékhez lesz kötve V -ben is.

Példák:

$$[x := 5, y := 7] + [x := 1, z := 3] = [x := 1, y := 7, z := 3]$$

$$[x := 1, z := 3] + [x := 5, y := 7] = [x := 5, y := 7, z := 3]$$

Kifejezések végrehajtása

Feltesszük, hogy a kifejezéseket a V környezetben hajtjuk végre.

1. Ha egy kifejezés egyetlen állandóból áll, akkor a kifejezés végrehajtása ennek az állandónak az értékét adja eredményül.
2. Ha egy kifejezés egyetlen azonosítóból áll, akkor a kifejezés végrehajtása ennek az azonosítónak a V -beli értékét adja eredményül.
3. Ha egy kifejezés a $\circ e$ operátoralkalmazásból áll, ahol \circ egy monadikus operátor, akkor a kifejezés végrehajtása az e részkifejezés V -beli értékének meghatározásával kezdődik. Ha ez a v értéket szolgáltatja, akkor a kifejezés végrehajtása a $\circ v$ operátoralkalmazás által szolgáltatott értéket adja eredményül.
4. Ha egy kifejezés az $e_1 \circ e_2$ operátoralkalmazásból áll, ahol \circ egy diadikus operátor, akkor a kifejezés végrehajtása az e_1 és e_2 részkifejezések V -beli értékének meghatározásával kezdődik. Ha ezek a v_1 és v_2 értékeket szolgáltatják, akkor a kifejezés végrehajtása a $v_1 \circ v_2$ operátoralkalmazás által szolgáltatott értéket adja eredményül.

5. Ha egy kifejezés a $\#k e$ projekcióból áll, akkor a kifejezés végrehajtása az e részkifejezés V -beli értékének meghatározásával kezdődik. Ha ennek az eredménye egy legalább k elemű ennes, akkor a kifejezés végrehajtása az ennes k -adik tagjának értékét adja eredményül.
6. Ha egy kifejezés az $e_1 e_2$ eljárásalkalmazásból áll, akkor a kifejezés végrehajtása az e_1 és e_2 részkifejezések V -beli értékének meghatározásával kezdődik. Ha ezek a p eljárást és a v értéket szolgáltatják, akkor a kifejezés végrehajtása a $p v$ eljárás hívás végrehajtásával kapott értéket adja eredményül.
7. Ha egy kifejezés az $\text{if } e_1 \text{ then } e_2 \text{ else } e_3$ feltételes kifejezésből áll, akkor a kifejezés végrehajtása az e_1 feltétel V -beli értékének meghatározásával kezdődik. A kifejezés végrehajtása, ha az e_1 végrehajtása a `true` értéket szolgáltatja, az e_2 következmény, ha pedig a `false` értéket szolgáltatja, az e_3 alternatíva végrehajtásával folytatódik. A kifejezés végrehajtásának az eredménye a következmény, illetve az alternatíva végrehajtásával kapott érték.
8. Ha egy kifejezés az (e_1, \dots, e_n) enneskifejezésből áll, akkor a kifejezés végrehajtása az e_1, \dots, e_n részkifejezések V -beli értékének meghatározásával kezdődik. Ha ezek a v_1, \dots, v_n értékeket szolgáltatják, akkor a típuskifejezés végrehajtásának eredménye a (v_1, \dots, v_n) ennes.

9. Ha egy kifejezés a let P in e end lokális deklarációt tartalmazó kifejezésből áll, akkor a kifejezés végrehajtása az P program V -beli végrehajtásával kezdődik. Ha ez a V' környezetet szolgáltatja, akkor a végrehajtás az e kifejezés V' -beli értékének meghatározásával folytatódik. A lokális deklarációt tartalmazó kifejezés végrehajtásának eredménye az így kapott érték.

Eljáráshívások végrehajtása

Egy $p = (\text{fun } fM = e, t, V)$ eljárásnak egy v értékkel történő meghívásakor a végrehajtás annak a V' környezetnek a meghatározásával kezdődik, amely az M mintában előforduló azonosítókat a v argumentum megfelelő értékeihez köti. Ezt követi az e eljárástörzs végrehajtása a $V + [f := p] + V'$ környezetben. Az eljáráshívás eredménye az így kapott érték lesz.

Az $[f := p]$ kötés teszi lehetővé a rekurzív eljáráshívásokat.

Deklarációk végrehajtása

Feltesszük, hogy a deklarációkat a V környezetben hajtjuk végre.

1. Egy $\text{val } M = e$ deklaráció végrehajtása az e kifejezés V -beli értékének meghatározásával kezdődik. Ha ez a v értéket szolgáltatja, akkor a végrehajtás annak a V' környezetnek a meghatározásával folytatódik, amely az M mintában előforduló azonosítókat a megfelelő v -beli értékekhez köti. A deklaráció végrehajtásának a $V + V'$ környezet az eredménye.
2. Egy $\text{fun } fM = e$ vagy $\text{fun } fM : t = e$ eljárásdeklaráció a

$$V + [f := (\text{fun } fM' = e, t', V')]$$

környezetet szolgáltatja, ahol

- (a) az M' az M -ből a típusok törlésével jön létre,
- (b) a t' a deklarált eljáráshoz rendelt típus,
- (c) a V' pedig az eljárásdeklaráció szabad azonosítóinak V -beli kötéseiből áll.

Programok végrehajtása

Feltesszük, hogy a programokat a V környezetben hajtjuk végre.

1. Ha a program üres, a végrehajtása a V környezetet adja eredményül.
2. Ha a program DP alakú, akkor a végrehajtása a D deklaráció V -beli végrehajtásával kezdődik. Ha ez a V' környezetet szolgáltatja, akkor a végrehajtás a P programrész V' -beli végrehajtásával folytatódik. A program végrehajtásának az így kapott környezet az eredménye.

A végrehajtás fázisai

1. Lexikális elemzés (lexical analysis): a karaktersorozat szavakra tagolása a szintaktikai szabályok alapján.
2. Szintaktikai elemzés (syntactic analysis, parsing): a szószorozat feldolgozása a szintaktikai szabályok alapján.
3. Szemantikai elemzés (semantic analysis, elaboration): a szintaktikailag helyes szószorozat feldolgozása a szemantikai szabályok alapján.
4. Végrehajtás (execution, evaluation) a szemantikai szabályok alapján.

Az elemzési fázisokban egy program *statikus*, a végrehajtási fázisban pedig a *dinamikus* aspektusairól beszélünk.

Egy programnyelv szintaxisa a lexikális és a szintaktikai elemzés alapja. Két részből áll: a lexikális szintaxisból és a mondatszintaxisból. A *lexikális szintaxis* a szavak, a *mondatszintaxis* a mondatok képzésének szabályait írja elő.

Egy programnyelv szemantikája a szemantikai elemzés és a végrehajtás alapja. Az előbbit a *statikus*, az utóbbit a *dinamikus* szemantika szabályozza.

Szintaktikai objektumoknak nevezzük a szavakat és mondatokat, *szemantikai objektumoknak* az értékeket és környezeteket. Különbséget teszünk *elemi* és *összetett* objektumok között is.

Vagyis egy programnyelv szintaxisa a nyelv külső megjelenését szabályozza, a szemantikája pedig a nyelv szintaktikai objektumainak a jelentésével foglalkozik.

Szemantikai egyenlőség

Két mondatot *szemantikailag egyenlőnek* mondunk, ha kívülről sem a statikus, sem a dinamikus szemantikájuk alapján nem lehet őket megkülönböztetni.

Például az $x+x$ és a $2*x$ kifejezések, valamint a $\text{fun } p(x:\text{int}, y:\text{int}) = x+y$

és a fun $p(y:int, x:int) = x+y$ eljárásdeklarációk szemantikailag egyenlők.

Egy programnyelv szemantikája a gyakorlati megvalósítás korlátjait, pl. a tár kapacitását vagy a számábrázolás tartományát nem veszi figyelembe. A szemantikai egyenlőséget is ilyen idealizált környezetben definiáljuk.

Idealizált környezetben például az $(x-y)*(x-y)$ és az $x*x - 2*x*y + y*y$ kifejezések szemantikailag egyenlők. Elég nagy számok esetén azonban a két kifejezés eltérően viselkedik a gyakorlatban:

```
val x = 999999999
```

```
val y = x
```

```
val z = (x-y)*(x-y)
```

```
z = 0 : int
```

```
val z = x*x - 2*x*y + y*y
```

```
!Uncaught exception
```

```
!Overflow
```