

Nyelvek típusrendszere (jegyzet)

Kaposi Ambrus
Eötvös Loránd Tudományegyetem
akaposi@inf.elte.hu

2016. október 17.

Tartalomjegyzék

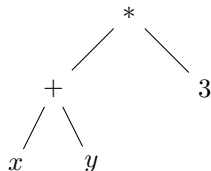
1. Bevezető	1
2. Induktív definíciók	2
2.1. Absztrakt szintaxisfák	3
2.2. Absztrakt kötések fák	5
2.3. Levezetési fák	8
3. Számok és szövegek	10
3.1. Szintaxis	11
3.2. Típusrendszer	11
3.3. Operációs szemantika átíró rendszerrel	16
3.4. Típusrendszer és szemantika kapcsolata	18
3.5. Futási idejű hibák	19
4. Függvények	20
4.1. Elsőrendű függvények	20
4.2. Magasabbrendű függvények	22
5. Véges adattípusok	23
5.1. Szorzat típusok	23
5.1.1. Nulláris és bináris szorzat típus	23
5.1.2. Véges szorzat típusok	24
5.2. Összeg típusok	24
5.3. Izomorfizmusok	24

1. Bevezető

Típusrendszerek helye a fordítóprogramokban:

1. Lexikális elemző (lexer): karakterek listájából tokenek listáját készíti, pl.
"(x + y) * 3"-ból `brOpen var[x] op[+] var[y] brClose op[*] const[3]`.

2. Szintaktikus elemző (parser): a tokenek listájából absztrakt szintaxisfát épít, az előbbi példából a következőt.



3. Típusellenőrzés: megnézi, hogy a szintaxisfa megfelel -e bizonyos szabályoknak (a típusrendszernek), pl. szükséges, hogy x és y numerikus típusú legyen. Általánosságban, nem enged át értelmetlen programokat. Mi ezzel a résszel foglalkozunk.
4. Kódgenerálás: ha a típusellenőrző átengedte a szintaxisfát, akkor ezt lefordítjuk a gép számára érthető kóddá.

Típusrendszerek és logika:

- Mi az, hogy számítás? Erre egy válasz a lambda kalkulus (alternatíva a Turing-gépekre), amely a Lisp programozási nyelv alapja. A különböző típusrendszereket azért fejlesztették ki, hogy megszorítsák a programokat a jó programokra.
- Mi az, hogy bizonyítás? Erre válaszol a logika az ítéletlogikával, predikátumkalkulussal.
- Típuselmélet: egy olyan rendszer, mely a két nézőpontot egyesíti. Ez egy típusrendszer, melynek speciális eseteit fogjuk tanulni. Megmutatja, hogy az intuitívan hasonlóan viselkedő fogalmak (pl. matematikai függvények, logikai következtetés, függvény típusú program, univerzális kvantor) valójában ugyanazok. Elvezet az egyenlőség egy új nézőpontjára, ami egy új kapcsolatot nyit a homotópia-elmélet és a típuselmélet között.

A magas kifejezőerejű típusrendszerekben már nem az a kérdés, hogy egy megírt program típusozható -e. A nézőpont megfordul: először adjuk meg a típust (specifikáljuk, hogy mit csinálhat a program), majd a típus alapján megírjuk a programot (ez akár automatikus is lehet, ha a típus eléggé megszorítja a megírható programok halmazát).

Két könyv alapján haladunk, de nem szükséges egyik sem a tananyag elsajátításához.

- Robert Harper. Practical Foundations for Programming Languages. Cambridge University Press, 2012.
- Csörnyei Zoltán. Bevezetés a típusrendszerek elméletébe. ELTE Eötvös Kiadó, 2012.

2. Induktív definíciók

Harper könyv: I. rész.

2.1. Absztrakt szintaxisfák

Absztrakt szintaxisfának (AST, abstract syntax tree) nevezzük az olyan fát, melyek leveleinél *változók* vannak, közbenső pontjaikon pedig *operátorok*. Például a természetes szám kifejezések és az ezekből és összeadásból álló kifejezések AST-it az alábbi definíciókkal adhatjuk meg.

$$\begin{aligned} n, n', \dots \in \text{Nat} &::= i \mid \text{zero} \mid \text{suc } n \\ e, e', \dots \in \text{Exp} &::= x \mid \text{num } n \mid e + e' \end{aligned}$$

Nat-ot és Exp-et *fajtának* nevezzük. A Nat fajtájú AST-eket n -el, n' -vel stb. jelöljük. Nat fajtájú AST lehet egy i változó, vagy létre tudjuk hozni a nulláris zero operátorral (aritása $()\text{Nat}$) vagy az unáris suc operátorral (aritása $(\text{Nat})\text{Nat}$). n egy tetszőleges Nat fajtájú AST-t jelöl, míg i maga egy Nat fajtájú AST, mely egy darab változóból áll.

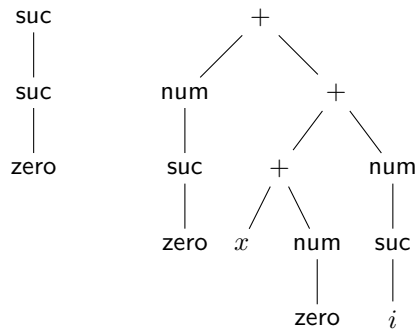
Az Exp fajtájú AST-eket e -vel és ennek vesszőzött változataival jelöljük, az Exp fajtájú változókat x -el jelöljük. Exp fajtájú AST-t egy unáris operátorral (num, aritása $(\text{Nat})\text{Exp}$) és egy bináris operátorral (+, aritása $(\text{Exp}, \text{Exp})\text{Exp}$) tudunk létrehozni. A num operátorral Nat fajtájú AST-eket tudunk kifejezésekbe beágyazni.

Minden fajtához változóknak egy külön halmaza tartozik, ezért jelöljük őket különböző betűkkel. A változók halmaza végtelen (mindig tudunk *friss* változót kapni, olyat, amelyet még sehol nem használtunk) és eldönthető, hogy két változó egyenlő-e. Az előbbi két fajtához tartozó változók halmazát így adhatjuk meg.

$$\begin{aligned} i, i', i_1 \dots &\in \text{Var}_{\text{Nat}} \\ x, x', x_1, \dots &\in \text{Var}_{\text{Exp}} \end{aligned}$$

A metaváltozókat (n, n', e, e' stb.) megkülönböztetjük a kifejezésekben szereplő változóktól, melyek Var_{Nat} , Var_{Exp} elemei. A metaváltozók a metanyelvünkben használt változók, a metanyelv az a nyelv, amiben ezek a mondatok íródnak.

Az AST-eket lerajzolhatjuk, pl. $\text{suc}(\text{suc zero})$ és $\text{num}(\text{suc zero}) + ((x + \text{num zero}) + \text{num}(\text{suc } i))$.



Az $x + \text{num zero}$ *részfája* az $(x + \text{num zero}) + \text{num}(\text{suc } i)$ AST-nek, ami pedig *részfája* a teljes AST-nek.

Az olyan AST-eket, melyek nem tartalmaznak változót, *zártak* nevezzük, egyébként *nyíltak*. A változók értékét *helyettesítéssel* (substitution) adhatjuk meg. Ha a egy A fajtájú AST, x pedig egy A fajtájú változó, t pedig tetszőleges fajtájú AST, akkor $t[x \mapsto a]$ -t úgy kapjuk meg t -ből, hogy x összes előfordulása

helyére a -t helyettesítünk. A helyettesítést az alábbi módon adjuk meg (f egy n -paraméteres operátor).

$$\begin{aligned} x[x \mapsto a] &:= a \\ y[x \mapsto a] &:= y && \text{feltéve, hogy } x \neq y \\ (f\ t_1 \dots t_n)[x \mapsto a] &:= f\ (t_1[x \mapsto a]) \dots (t_n[x \mapsto a]) \end{aligned}$$

Néhány példa:

$$\begin{aligned} (x + \text{num zero})[x \mapsto \text{num (suc zero)}] &= \text{num (suc zero)} + \text{num zero} \\ (x + x)[x \mapsto x' + \text{num zero}] &= (x' + \text{num zero}) + (x' + \text{num zero}) \\ (x + \text{num (suc } i)) [i \mapsto \text{zero}] &= x + \text{num (suc zero)} \end{aligned}$$

Megjegyezzük, hogy a $\text{num (suc zero)} + \text{num (suc zero)}$ egy formális kifejezés, amelynek a jelentését (szemantikáját) még nem adtuk meg. Fontos, hogy emiatt ne keverjük össze az $1 + 1 = 2$ természetes számmal. Pl. nem igaz, hogy $\text{num (suc zero)} + \text{num (suc zero)} = \text{num (suc (suc zero))}$, ez két különböző AST.

Ha szeretnénk bizonyítani, hogy egy állítás az összes adott fajtájú AST-re igaz, elég megmutatni, hogy a változókra igaz az állítás, és az összes operátor megtartja az állítást. Pl. ha minden $n \in \text{Nat}$ -ra szeretnénk belátni, hogy $\mathcal{P}(n)$, akkor elég azt belátni, hogy $\mathcal{P}(i)$ minden i -re, hogy $\mathcal{P}(\text{zero})$ és hogy minden m -re $\mathcal{P}(m)$ -ből következik $\mathcal{P}(\text{suc } m)$. Ha minden $e \in \text{Exp}$ -re szeretnénk $\mathcal{Q}(e)$ -t belátni, elég azt belátni, hogy minden x -re $\mathcal{Q}(x)$, hogy $n \in \text{Nat}$ -ra igaz $\mathcal{Q}(\text{num } n)$ és ha igaz $\mathcal{Q}(e)$ és $\mathcal{Q}(e')$, akkor igaz $\mathcal{Q}(e + e')$ is. Ezt az érvelési formát szerkezeti *indukciónak* nevezzük (structural induction).

Ha egy függvényt szeretnénk megadni, ami az összes adott fajtájú AST-n működik, akkor hasonlóképp azt kell meghatározni, hogy az egyes operátorokon hogyan működik a függvény (felhasználva azt, hogy mi a függvény eredménye az operátorok paraméterein). Ezt a megadási módot szerkezeti *rekurzió*-nak hívjuk (structural recursion).

2.1. Feladat. Végezzük el a következő helyettesítéseket:

$$\begin{aligned} ((x + x')[x \mapsto x']) [x' \mapsto x] \\ (x + x)[x \mapsto x' + x] \\ (\text{suc (suc } i)) [i \mapsto \text{suc } i'] \\ (\text{num (suc } i)) [x \mapsto \text{num zero}] \end{aligned}$$

2.2. Feladat. Adjuk meg a Nat -ok listájának fajtáját az operátorok aritásával együtt.

$$xs \in \text{List}_{\text{Nat}} ::= ?$$

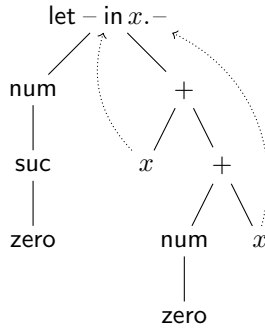
További információ:

- A különböző fajtájú AST-k halmazait finomíthatjuk aszerint, hogy milyen változók vannak bennük. Ha egy Exp fajtájú e AST-ben x, x' a szabad változók, azt mondjuk, hogy $e \in \text{Exp}_{x, x'}$ halmazban van. Ekkor pl. $e + x'' \in \text{Exp}_{x, x', x''}$ és $e[x \mapsto \text{num zero}] \in \text{Exp}_{x'}$. Általános esetben szükségünk van minden fajtához egy különböző változó-halmazra, így ha $a \in A_{X_A \dots X_B}$ és $b \in B_{Y_A, y \dots Y_B}$, akkor $b[y \mapsto a] \in B_{X_A \cup Y_A \dots X_B \cup Y_B}$.

- Az ABT-ket polinomiális funktorok (polynomial functor, container) szabad monádjaként (free monad) adjuk meg. Ezzel biztosítjuk, hogy pl. minden szintaxisfa véges.

2.2. Absztrakt kötési fák

Az *absztrakt kötési fák* (ABT, abstract binding tree) az AST-khez hasonló, de változót *kötő* operátorok is szerepelhetnek benne. Ilyen például a $\text{let } e \text{ in } x.e'$, mely pl. azt fejezheti ki, hogy e' -ben az x előfordulásai e -t jelentenek (ez a let kifejezések egy lehetséges szemantikája, de ebben a fejezetben csak szintaxisal foglalkozunk, emiatt nem igaz, hogy $\text{let } e \text{ in } x.x + x = e + e$). Azt mondjuk, hogy az x változó kötve van az e' kifejezésben. A let operátor aritását $(\text{Exp}, \text{Exp}.\text{Exp})\text{Exp}$ -el jelöljük, az operátor második paraméterében köt egy Exp fajtájú változót. Pl. $\text{let num (suc zero) in } x.x + (\text{num zero} + x)$ kifejezésben a $+$ operátor x paraméterei a kötött x -re vonatkoznak. Ezt a következőképp ábrázolhatjuk. A felfele mutató szaggatott nyilak mutatják, hogy az x változók melyik kötésre mutatnak. A pont után szereplő $x + (\text{num zero} + x)$ részkifejezést az x változó *hatáskörének* nevezzük.



A let-tel kiegészített Exp fajtájú ABT-ket a következő jelöléssel adjuk meg. A pont azelőtt a paraméter előtt van, amiben kötjük a változót.

$$e, e' \in \text{Exp} ::= x \mid \text{num } n \mid e + e' \mid \text{let } e \text{ in } x.e'$$

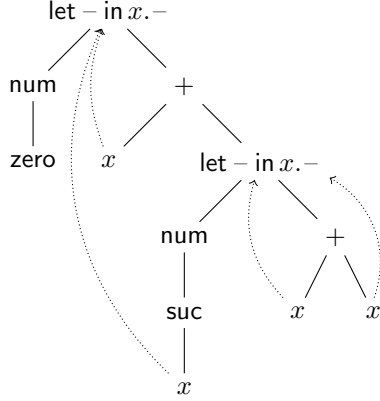
A $\text{let num zero in } x.x + x$ ABT $x + x$ részfája tartalmaz egy *szabad változót*, az x -et, emiatt az $x + x$ ABT nyílt. Mivel $x + x$ -ben csak az x a szabad változó, a kötés zárttá teszi a kifejezést, tehát a $\text{let zero in } x.x + x$ ABT zárt. A kötések hatásköre a lehető legtovább tart, emiatt $\text{let } e \text{ in } x.(e_0 + e_1) = \text{let } e \text{ in } x.e_0 + e_1 \neq (\text{let } e \text{ in } x.e_0) + e_1$.

Az alábbi Exp fajtájú ABT-k kötött változóit aláhúztuk, szabad változóit felülhúztuk.

$\text{let num zero in } x.\underline{x} + \underline{x}$
 $\text{let } \bar{y} \text{ in } x.\underline{x} + \underline{x}$
 $\text{let } \bar{y} \text{ in } x.\underline{x} + \text{let } \underline{x} \text{ in } z.\underline{z}$
 $\text{let } \bar{y} \text{ in } x.\underline{x} + \text{let } \bar{z} \text{ in } z.\underline{z}$

A kötött változók csak pozíciókra mutatnak, a nevük nem érdekes. Például a $\text{let num zero in } x.x + x$ és a $\text{let num zero in } y.y + y$ ABT-k megegyeznek (α -konvertálhatónak vagy α -ekvivalensnek szokás őket nevezni). A szabad változókra ez nem igaz, pl. $x + x \neq y + y$.

Ha többször ugyanazt a változót kötjük egy ABT-ben, az újabb kötés *elfedi* az előzőt. Pl. $\text{let num zero in } x.x + (\text{let num (suc } x) \text{ in } x.x + x)$ -ben az $x + x$ -ben levő x -ek a második kötésre (ahol $\text{num (suc } x)$ -et adtunk meg) mutat (a $\text{num (suc } x)$ -ben levő x viszont az első kötésre mutat).



Az elfedés megszüntethető a változónevek átnevezésével: $\text{let num zero in } y.y + (\text{let num (suc } y) \text{ in } x.e)$. Ebben az ABT-ben már hivatkozhatunk az e részében az x -re is meg a külső y -ra is.

Helyettesíteni tudunk ABT-kben is, pl. szeretnénk a következő egyenlőségeket.

$$\begin{aligned} (\text{let } x \text{ in } x'.x' + x'')[x'' \mapsto \text{num zero}] &= \text{let } x \text{ in } x'.x' + \text{num zero} \\ (\text{let } x \text{ in } x'.x' + x'')[x' \mapsto \text{num zero}] &= \text{let } x \text{ in } x'.x' + x'' \\ (\text{let } x \text{ in } x'.x' + x'')[x'' \mapsto x'] &= \text{let } x \text{ in } x'''.x''' + x' \end{aligned}$$

Az első esetben egyszerűen behelyettesítünk az x' -t kötő művelet alatt. A második esetben, mivel a kötés elfedi az x' változót, a kötés hatáskörében levő x' -k mind a kötésre vonatkoznak, ezért nem történik semmi. A harmadik eset érdekesebb: itt azért, hogy a kötés alá mentünk, naivan csak lecserélnénk az x'' -t x' -re, de ezzel az x' kötötté válna, és nem a „külső” x' -re, hanem a kötöttre vonatkozna, megváltoztatva ezzel az ABT jelentését. Emiatt a kötésre egy másik, még nem használt változót, x''' -t használjuk.

Általánosságban a következőképp tudjuk megadni a helyettesítést egy f operátorra, mely az első paraméterében köt¹.

$$(f(y.t) t_1 \dots t_n)[x \mapsto a] := f\left(\left(z.(t[y \mapsto z])[x \mapsto a]\right)(t_1[x \mapsto a]) \dots (t_n[x \mapsto a])\right)$$

(ahol z friss változónév)

A biztonság kedvéért (lásd a fenti harmadik példát), a kötött változót átnevezük egy friss z változóra, és csak ezután a helyettesítés után helyettesítjük a megmaradt x -eket a -val.

Ha szeretnénk szerkezeti indukcióval egy \mathcal{Q} állítást a let -tel kiegészített Exp ABT-kről bizonyítani, a következőket kell belátnunk:

- minden x változóra $\mathcal{Q}(x)$,
- minden $n \in \text{Nat}$ -ra $\mathcal{Q}(n)$,

¹ Ha több paraméterben van kötés, azt is hasonlóan lehet megadni.

- minden $e, e' \in \text{Exp-re}$, ha $\mathcal{Q}(e)$ és $\mathcal{Q}(e')$, akkor $\mathcal{Q}(e + e')$,
- minden $e, e' \in \text{Exp-re}$ és x változóra, ha $\mathcal{Q}(e)$ és $\mathcal{Q}(e')$, akkor $\mathcal{Q}(\text{let } e \text{ in } x.e')$.

2.3. Feladat. *Húzd alá a kötött változókat és fölé a szabad változókat! A kötött változóknál rajzolj egy nyilat, hogy melyik kötésre mutatnak!*

```

x + num i
let x in x.num i + x
let x in x.x' + let x in x'.x' + x
let x in x.x + let x in x.x + x
let x in x.(let x in x'.x'' + x') + let x' in x'.x' + x''

```

2.4. Feladat. *Lehet -e egy $i \in \text{Var}_{\text{Nat}}$ egy Exp fajtájú ABT-ben kötött?*

2.5. Feladat. *Adj algoritmust arra, hogy két ABT mikor egyenlő általános esetben. Az érdekes rész annak eldöntése, hogy egy $f\ x.t\ t_1 \dots t_n$ alakú és egy $f\ x'.t'\ t'_1 \dots t'_n$ alakú ABT egyenlő -e.*

2.6. Feladat. *Végezd el a következő helyettesítéseket!*

```

(let x in x.x + x)[x ↦ num zero]
(let x' in x.x' + x)[x' ↦ num zero]
(let x' in x.x' + x')[x' ↦ x]

```

2.7. Feladat. *Döntsd el, hogy a következő Exp fajtájú ABT-k megegyeznek -e!*

```

x + num i  $\stackrel{?}{=}$  x + num i'
let x in x.num i + x  $\stackrel{?}{=}$  let x in x'.num i + x'
let x in x.num i + x  $\stackrel{?}{=}$  let x in x.num i + x'
let x in x.x' + let x in x'.x' + x  $\stackrel{?}{=}$  let x in x'.x + let x' in x.x + x'
let x in x.x' + let x in x'.x' + x  $\stackrel{?}{=}$  let x in x''.x' + let x'' in x.x + x''
let x in x.x + let x in x.x + x  $\stackrel{?}{=}$  let x in x'.x' + let x' in x'.x' + x'

```

2.8. Feladat. *Írj minél több zárt ABT-t az alább megadott fajtában. d aritása (A.A)A, g aritása (A, A)A.*

```

a ∈ A ::= y | d y.a | g a a
y, y', ... ∈ VarA

```

További információ:

- Az α -ekvivalencia legegyszerűbb implementációja a De Bruijn indexek használata. Változónevek helyett természetes számokat használunk, melyek azt mutatják, hogy hányadik kötésre mutat a változó. Pl. $d\ y.d\ y'.y$ helyett $d\ (d\ 1)\text{-et}$ írunk (0 mutatna a legközelebbi kötésre).

2.3. Levezetési fák

Ítéleteket ABT-kről mondunk. Néhány példa ítéletekre és a lehetséges jelentésükre:

$n \text{ nat}$	n egy természetes szám
$n + n' \text{ is } n''$	az n és n' természetes számok összege n''
$e \text{ hasHeight } n$	az e bináris fának n a magassága
$e : \tau$	az e kifejezésnek τ a típusa
$e \mapsto e'$	az e kifejezés e' -re redukálódik

Az ítéletek *levezetési szabályokkal* vezethetők le. A levezetési szabályok általános formája az alábbi. J_1, \dots, J_n -t feltételeknek, J -t következménynek nevezzük.

$$\frac{J_1 \quad \dots \quad J_n}{J}$$

Az $n + n' \text{ is } n''$ ítélethez például az alábbi kettő levezetési szabályt adjuk meg.

$$\overline{\text{zero} + n' \text{ is } n'} \quad (2.1)$$

$$\frac{n + n' \text{ is } n''}{\text{suc } n + n' \text{ is } \text{suc } n''} \quad (2.2)$$

Az első szabály azt fejezi ki, hogy nullát hozzáadva bármilyen számhoz ugyanazt a számot kapjuk. A második szabály azt fejezi ki, hogy ha tudjuk két szám összegét, akkor az első rákövetkezőjének és a másodiknak az összege az összeg rákövetkezője lesz. Az n, n', n'' metaváltozók bármilyen Nat fajtájú ABT-t jelenthetnek. A 2.1 szabályban fontos, hogy a két n' mindig ugyanaz kell, hogy legyen. Megjegyezzük, hogy ebben az ítéletben a $+$ -nak semmi köze sincsen az Exp fajtájú ABT-kben szereplő $+$ operátorhoz, csak véletlenül ugyanazzal a karakterrel jelöljük.

A levezetési szabályok *levezetési fává* (levezetéssé) kombinálhatók. Pl. azt, hogy $2 + 1 = 3$, a következőképp tudjuk levezetni.

$$\frac{\overline{\text{zero} + \text{suc zero is suc zero}} \quad (2.1)}{\frac{(\text{suc zero}) + \text{suc zero is suc (suc zero)}}{\text{suc (suc zero) + suc zero is suc (suc (suc zero))}} \quad (2.2)} \quad (2.2)$$

A levezetési fa gyökerénél van, amit levezettünk, és mindegyik lépésben valamelyik szabályt alkalmaztuk: az alkalmazott szabály száma a vízszintes vonal mellé van írva.

Az olyan levezetési szabályokat, melyeknek nincs feltétele, *axiómának* nevezzük. A levezetési fák leveleinél mindig axiómák vannak.

Az $n \text{ nat}$ ítélet azt fejezi ki, hogy n egy természetes szám. A következő szabályokkal tudjuk levezetni.

$$\overline{\text{zero nat}} \quad (2.3)$$

$$\frac{n \text{ nat}}{\text{suc } n \text{ nat}} \quad (2.4)$$

Ezek azt fejezik ki, hogy a 0 természetes szám, és bármely természetes szám rákövetkezője is az. N.b. azt nem tudjuk levezetni, hogy $i \text{ nat}$ egy i változóra. A természetes számokra gondolhatunk úgy, mint a zárt Nat fajtájú ABT-kre.

A következő levezetési szabályok azt fejezik ki, hogy az Exp fajtájú AST kiegyensúlyozott, és magassága n . Az ítélet általános alakja $e \text{ isBalanced } n$.

$$\overline{x \text{ isBalanced zero}} \quad (2.5)$$

$$\overline{\text{num } n \text{ isBalanced zero}} \quad (2.6)$$

$$\frac{e \text{ isBalanced } n \quad e' \text{ isBalanced } n}{e + e' \text{ isBalanced suc } n} \quad (2.7)$$

Egy példa levezetés.

$$\frac{\overline{x' \text{ isBalanced zero}} \quad \overline{x \text{ isBalanced zero}} \quad \overline{x' \text{ isBalanced zero}} \quad \overline{\text{num zero isBalanced zero}}}{\frac{x + x \text{ isBalanced suc zero} \quad x' + \text{num } i \text{ isBalanced suc zero}}{(x + x) + (x' + \text{num } i) \text{ isBalanced suc (suc zero)}}$$

Ha valamit be szeretnénk bizonyítani minden levezethető ítéletről, ehhez a szabályok szerinti szerkezeti indukciót használhatjuk. Azt szeretnénk belátni, hogy ha J levezethető, akkor $\mathcal{P}(J)$ igaz. Ebben az esetben elég belátnunk azt, hogy minden szabályra, melynek feltételei J_1, \dots, J_n és következménye J , ha $\mathcal{P}(J_1), \dots, \mathcal{P}(J_n)$ mind teljesül, akkor $\mathcal{P}(J)$ is.

A szerkezeti indukció konkrét használatára mutatunk két példát.

A fenti 2.1 szabály alapján bármely $n \in \text{Nat}$ -ra le tudjuk vezetni, hogy $\text{zero} + n$ is n . Megmutatjuk a másik irányt.

2.9. Lemma. *Ha n egy természetes szám, akkor $n + \text{zero}$ is n .*

Bizonyítás. Indukció a természetes számok levezetésén. A fenti P -t úgy választjuk meg, hogy $P(n \text{ nat}) := n + \text{zero} \text{ is } n$. Ha a 2.3 szabályt használtuk, akkor $P(\text{zero nat}) = \text{zero} + \text{zero} \text{ is zero}$ -t kell bizonyítanunk, ezt megtesszük a 2.1 szabállyal. Ha a 2.4 szabályt használtuk, akkor az indukciós hipotézis azt mondja, hogy $P(n \text{ nat}) = n + \text{zero} \text{ is } n$, nekünk pedig azt kell bizonyítani, hogy $P(\text{suc } n \text{ nat}) = \text{suc } n + \text{zero} \text{ is suc } n$. A 2.2 szabályt használjuk, a feltételét az indukciós feltevésünk adja meg. \square

Második példaként bebizonyítjuk, hogy ha van egy kiegyensúlyozott fánk, és ebbe behelyettesítünk egy 0 magasságú fát, akkor a kapott fa is kiegyensúlyozott lesz.

2.10. Lemma. *Ha $e_1 \text{ isBalanced zero}$ és $e \text{ isBalanced } n$, akkor bármely x_1 -re $e[x_1 \mapsto e_1] \text{ isBalanced } n$.*

Bizonyítás. $e \text{ isBalanced } n$ levezetése szerinti indukcióval bizonyítunk, tehát $P(e \text{ isBalanced } n) = e[x_1 \mapsto e_1] \text{ isBalanced } n$. A következő eseteket kell ellenőriznünk.

- Ha a 2.5 szabályt használtuk $e \text{ isBalanced } n$ levezetésére, akkor azt kell belátnunk, hogy $x[x_1 \mapsto e_1] \text{ isBalanced zero}$. Ha $x = x_1$, akkor ez azzal egyezik meg, hogy $e'_1 \text{ isBalanced zero}$, ezt pedig tudjuk. Ha $x \neq x_1$, akkor a 2.5 szabályt használjuk újra.

- Ha a 2.6 szabályt használtuk, akkor azt kell belátnunk, hogy $(\text{num } n)[x_1 \mapsto e_1]$ `isBalanced zero`, viszont a helyettesítés itt nem végez semmit, tehát a 2.6 szabályt újra alkalmazva megkapjuk a kívánt eredményt.
- Ha a 2.7 szabályt alkalmaztuk, akkor az indukciós feltevésekből tudjuk, hogy $e[x_1 \mapsto e_1]$ `isBalanced n` és $e'[x_1 \mapsto e_1]$ `isBalanced n` , és azt szeretnénk belátni, hogy $(e + e')[x_1 \mapsto e_1]$ `isBalanced $\text{succ } n$` , de a helyettesítés definíciója alapján ez megegyezik $e[x_1 \mapsto e_1] + e'[x_1 \mapsto e_1]$ `isBalanced $\text{succ } n$` -al, amit pedig a 2.7 szabály alapján látunk.

□

2.11. Feladat. Adjuk meg a $\max n n' = n''$ ítélet levezetési szabályait. Az ítélet azt fejezi ki, hogy n és n' `Nat`-beli ABT-k maximuma n'' .

2.12. Feladat. Ennek segítségével adjuk meg a `e hasHeight n` ítélet levezetési szabályait.

2.13. Feladat. Adjuk meg a `isEven n` és `isOdd n` ítéletek levezetési szabályait, melyek azt fejezik ki, hogy n páros ill. páratlan szám. A levezetési szabályok hivatkozhatnak egymásra.

2.14. Feladat. Igaz-e, hogy bármely két zárt természetes számra levezethető, hogy mennyi azok összege a természetes számok összegének két levezetési szabályával?

További információ:

- Az AST-k, ABT-k és a levezetési fák mind induktív definíciók, típuselméletben induktív családokként formalizálhatók (inductive families). A szerkezeti indukció elvét ebben az esetben az eliminátor fejezi ki.

3. Számok és szövegek

Harper könyv: II. rész.

Ebben a fejezetben egy egyszerű nyelvet tanulmányozunk, mely számokból és szövegekből álló kifejezéseket tartalmaz. A nyelv szintaxisa az ABT-k definíciója, melyekből a nyelv áll. A típusrendszer az összes lehetséges ABT-t megszorítja értelmes ABT-kre. A szemantika pedig a nyelv jelentését adja meg: azt, hogy futási időben mi történik az ABT-ekkel.

3.1. Szintaxis

A szintaxis két fajtából áll, a típusokból és a kifejezésekből.

$$\tau, \tau', \dots \in \text{Ty} ::= \text{int} \quad \text{egész számok típusa} \quad (3.1)$$

$$| \text{str} \quad \text{szövegek típusa}$$

$$e, e', \dots \in \text{Exp} ::= x \quad \text{változó} \quad (3.2)$$

$$| n \quad \text{egész szám beágyazása}$$

$$| "s" \quad \text{szöveg beágyazása}$$

$$| e + e' \quad \text{összeadás}$$

$$| e - e' \quad \text{kivonás}$$

$$| e \bullet e' \quad \text{összefűzés}$$

$$| |e| \quad \text{hossz}$$

$$| \text{let } e \text{ in } x.e' \quad \text{definíció}$$

A típusok kétfélék lehetnek, `int` és `str`, típusváltozókat nem engedélyezünk. A kifejezések mellé írtuk a jelentésüket, általában így gondolunk ezekre a kifejezésekre, hogy ezek számokat, szövegeket reprezentálnak. De fontos, hogy ezek nem tényleges számok, csak azok szintaktikus reprezentációi. A szintaxis csak egy formális dolog, karakterek sorozata (pontosabban egy ABT), jelentését majd a szemantika adja meg.

Az `Exp` fajtájú változókat x, y, z és ezek indexelt változatai jelölik.

A két beágyazás operátor paramétere egy egész szám ill. egy szöveg, ezeket adottnak tételezzük fel, tehát a metanyelvünkben léteznek egész számok, pl. `0, 1, -2` és szövegek, pl. `hello`. A (jelöletlen) egész szám beágyazása operátorral a nyelvünkben `Exp` fajtájú kifejezés a `0, 1` és a `-2` is, míg az idézőjelekkel jelölt szöveg beágyazása operátorral `Exp` fajtájú kifejezés a `"hello"`. A `let` operátorral tudunk definíciókat írni ebben a nyelvben, pl. `let "ello" in x."h" • x • "b" • x`.

Az operátorok aritásai a szintaxis definíciójából leolvashatók, pl. a (jelöletlen) beágyazás operátor aritása $(\mathbb{Z})\text{Exp}$, $|-$ aritása $(\text{Exp})\text{Exp}$, `let` aritása $(\text{Exp}, \text{Exp}, \text{Exp})\text{Exp}$.

3.1. Feladat. Írjuk fel az összes operátor aritását!

3.2. Típusrendszer

A típusrendszer megkorlátozza a leírható kifejezéseket azzal a céllal, hogy az értelmetlen kifejezéseket kiszűrje. Pl. a $|3|$ kifejezést ki szeretnénk szűrni, mert szeretnénk, hogy a `hossz` operátort csak szövegekre lehessen alkalmazni. Az, hogy pontosan milyen hibákat szűr ki a típusrendszer, nincs egységesen meghatározva, ez a típusrendszer tervezőjén múlik.

Hogy a típusrendszert le tudjuk írni, szükségünk van még a környezetek (context) fajtájára. Egy környezet egy változókból és típusokból álló lista.

$$\Gamma, \Gamma', \dots \in \text{Con} ::= \cdot \mid \Gamma, x : \tau \quad (3.3)$$

A célunk a környezettel az, hogy megadja a kifejezésekben levő szabad változók típusait. A típusozási ítélet $\Gamma \vdash e : \tau$ formájú lesz, ami azt mondja, hogy az e kifejezésnek τ típusa van, feltéve, hogy a szabad változói típusai a Γ által megadottak.

Emiatt bevezetünk egy megszorítást a környezetekre: egy változó csak egyszer szerepelhet. Először is megadunk egy függvényt, mely kiszámítja a környezet változóit tartalmazó halmazt.

$$\begin{aligned}\text{dom}(\cdot) &:= \{\} \\ \text{dom}(\Gamma, x : \tau) &:= \{x\} \cup \text{dom}(\Gamma)\end{aligned}\tag{3.4}$$

A $\Gamma \text{ wf}$ ítélet azt fejezi ki, hogy a Γ környezet jól formált.

$$\overline{\cdot \text{ wf}}\tag{3.5}$$

$$\frac{\Gamma \text{ wf} \quad x \notin \text{dom}(\Gamma)}{\Gamma, x : \tau \text{ wf}}\tag{3.6}$$

Ezután csak jól formált környezetekkel fogunk dolgozni.

Bevezetünk egy ítéletet, amely azt mondja, hogy egy változó-típus pár benne van egy környezetben.

$$\frac{\Gamma \text{ wf} \quad x \notin \text{dom}(\Gamma)}{(x : \tau) \in \Gamma, x : \tau}\tag{3.7}$$

$$\frac{(x : \tau) \in \Gamma \quad y \notin \text{dom}(\Gamma)}{(x : \tau) \in \Gamma, y : \tau'}\tag{3.8}$$

Az első szabály azt fejezi ki, hogy ha egy környezet utolsó alkotóeleme $x : \tau$, akkor ez természetesen szerepel a környezetben. Továbbá, ha egy környezetben $x : \tau$ szerepel, akkor egy y változóval kiegészített környezetben is szerepel.

A típusrendszerrel $\Gamma \vdash e : \tau$ formájú ítéleteket lehet levezetni, ami azt jelenti, hogy a Γ környezetben az e kifejezésnek τ típusa van. Úgy is gondolhatunk erre, hogy e egy program, melynek típusa τ és a program paraméterei és azok típusai Γ -ban vannak megadva. A levezetési szabályok a következők.

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x : \tau}\tag{3.9}$$

$$\frac{\Gamma \text{ wf}}{\Gamma \vdash n : \text{int}}\tag{3.10}$$

$$\frac{\Gamma \text{ wf}}{\Gamma \vdash "s" : \text{str}}\tag{3.11}$$

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}}\tag{3.12}$$

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 - e_2 : \text{int}}\tag{3.13}$$

$$\frac{\Gamma \vdash e_1 : \text{str} \quad \Gamma \vdash e_2 : \text{str}}{\Gamma \vdash e_1 \bullet e_2 : \text{str}}\tag{3.14}$$

$$\frac{\Gamma \vdash e : \text{str}}{\Gamma \vdash |e| : \text{int}}\tag{3.15}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2 \quad x \notin \text{dom}(\Gamma)}{\Gamma \vdash \text{let } e_1 \text{ in } x.e_2 : \tau_2}\tag{3.16}$$

Példa levezetés:

$$\begin{array}{c}
\frac{\frac{\frac{\cdot \text{wf} \quad 3.5}{\cdot \vdash \text{"a"} : \text{str}} \quad 3.11 \quad \frac{\frac{\frac{\frac{\cdot \text{wf} \quad 3.5}{(x : \text{str}) \in \cdot, x : \text{str}} \quad 3.7}{\cdot, x : \text{str} \vdash x : \text{str}} \quad 3.9}{\cdot, x : \text{str} \vdash |x| : \text{int}} \quad 3.15}{\cdot, x : \text{str} \vdash |x| + 2 : \text{int}} \quad 3.16}{\cdot \vdash \text{let "a" in } x. |x| + 2 : \text{int}} \quad 3.16 \\
\frac{\frac{\cdot \text{wf} \quad 3.5}{\cdot, x : \text{str wf}} \quad 3.6}{\cdot, x : \text{str} \vdash 2 : \text{int}} \quad 3.10
\end{array}$$

A $|3|$ kifejezés nem típusozható, mert csak a 3.15. szabállyal vezethető le $|e|$ alakú kifejezés, ez a szabály viszont azt követeli meg, hogy a 3 kifejezés str típusú legyen. Ezt azonban egyik szabállyal sem lehet levezetni.

Most tételeket fogunk kimondani a típusrendszerről. Fontos, hogy legyen intuíciónk arról, hogy miért igazak ezek a tételek. A bizonyítások általában egyszerű szerkezeti indukciók, de némely lépésben korábbi tételeket is felhasználunk. A bizonyítások megértéséhez feltétlenül szükséges, hogy ne csak elolvassuk őket, hanem magunk is levezessük őket papíron.

A típusrendszer nem triviális, tehát nem igaz, hogy minden kifejezés típusozható.

3.2. Lemma. *Van olyan e melyre nem létezik Γ és τ hogy $\Gamma \vdash e : \tau$.*

Bizonyítás. Például az előbbi $|3|$. □

Minden kifejezés maximum egyféleképpen típusozható (típusok unicitása).

3.3. Lemma. *Egy e kifejezéshez és Γ környezethez maximum egy τ típus tartozik, melyre $\Gamma \vdash e : \tau$.*

Bizonyítás. Azt látjuk be a $\Gamma \vdash e : \tau$ levezetése szerinti indukcióval, hogy másik $\tau' \neq \tau$ típusra nincs olyan szabály, mely levezetné $\Gamma \vdash e : \tau'$ -t. A változó esetén a szabály szerinti indukcióval belátjuk, hogy ha $(x : \tau) \in \Gamma$, akkor nem lehet, hogy $(x : \tau') \in \Gamma$ valamely $\tau' \neq \tau$ -ra. □

A típusrendszer ezenkívül szintaxisvezérelt: ez azt jelenti, hogy minden kifejezésformát pontosan egy szabály tud levezetni. Emiatt a típusozás megfordítható. A típus levezetési szabályok azt adják meg, hogy melyek az elégséges feltételek ahhoz, hogy egy kifejezés típusozható legyen. Pl. ahhoz, hogy $e_1 + e_2$ típusa int legyen, elég az, hogy e_1 és e_2 típusa int ugyanabban a környezetben. Mi meg tudjuk adni a szükséges feltételeket is (típusozás inverziója). Ez mutatja meg, hogyan tudunk egy típusellenőrző programot írni ehhez a típusrendszerhez.

3.4. Lemma. $\Gamma \vdash e : \tau$ levezethető. Ekkor, ha $e = e_1 + e_2$, akkor $\tau = \text{int}$, $\Gamma \vdash e_1 : \text{int}$ és $\Gamma \vdash e_2 : \text{int}$. Hasonlóképp az összes többi operátorra.

Bizonyítás. A levezetés szerinti indukcióval. □

A környezetben a változó-típus párok sorrendje nem számít ebben a típusrendszerben. Ezt fejezi ki a következő lemma.

3.5. Lemma. *Ha $\Gamma \vdash e : \tau$, akkor $\Gamma' \vdash e : \tau$, feltéve, hogy Γ' a Γ egy permutációja (ugyanazok a változó-típus párok, csak más sorrendben).*

Bizonyítás. A $\Gamma \vdash e : \tau$ levezetése szerinti indukcióval. A 3.10–3.15. szabályoknál csak az induktív hipotéziseket, majd a szabályt használjuk. A 3.16. szabály esetén azt szeretnénk belátni, hogy $\Gamma' \vdash \text{let } e_1 \text{ in } x.e_2 : \tau_2$, feltéve, hogy $\Gamma' \vdash e_1 : \tau_1$ és $\Gamma'' \vdash e_2 : \tau_2$, ha Γ'' a $\Gamma, x : \tau_1$ egy permutációja. Mivel Γ' a Γ egy permutációja, ezért a $\Gamma', x : \tau_1$ a $\Gamma, x : \tau_1$ permutációja, így ez az indukciós feltevés azt mondja, hogy $\Gamma', x : \tau_1 \vdash e_2 : \tau_2$. Ezt felhasználva a 3.16. szabály alapján megkapjuk, hogy $\Gamma' \vdash \text{let } e_1 \text{ in } x.e_2 : \tau_2$. A 3.9. szabály esetén azt látjuk be, hogy ha $(x : \tau) \in \Gamma$, és Γ' a Γ egy permutációja, akkor $(x : \tau) \in \Gamma'$. Itt egyszerűen annyiszor alkalmazzuk a 3.6 szabályt, ahányadik komponens $(x : \tau)$ hátulról Γ' -ben. \square

A típusrendszer egy további fontos tulajdonsága a gyengítési tulajdonság.

3.6. Lemma. *Ha $\Gamma \vdash e : \tau$ levezethető, és $y \notin \text{dom}(\Gamma)$, akkor $\Gamma, y : \tau' \vdash e : \tau$ is levezethető bármely τ' -re.*

Ez a tulajdonság azt fejezi ki, hogy egy jól típusozott kifejezést tetszőleges olyan környezetben használhatunk, amiben meg vannak adva a szabad változói. Tehát ha írunk egy típushelyes programot, és ezután további paramétereket adunk meg a programnak, akkor a program ugyanúgy típushelyes marad.

Bizonyítás. A levezetés szerinti indukcióval. Tehát $P(\Gamma \vdash e : \tau) = \Gamma, y : \tau' \vdash e : \tau$, ahol $y \notin \text{dom}(\Gamma)$. A változó esetében eggyel többször használjuk a 3.8. szabályt, a 3.10–3.11. szabályok esetén ugyanazeket a szabályokat alkalmazzuk más környezetekre, a 3.12–3.15. szabályok esetén az indukciós feltevést használjuk, majd magát a szabályt. A 3.16. szabály esetén az indukciós feltevés azt mondja, hogy $\Gamma, y : \tau' \vdash e_1 : \tau_1$ és $\Gamma, x : \tau_1, y : \tau' \vdash e_2 : \tau_2$, nekünk pedig azt kell belátnunk, hogy $\Gamma, y : \tau' \vdash \text{let } e_1 \text{ in } x.e_2 : \tau_2$. Mivel $\Gamma, y : \tau', x : \tau_1$ a $\Gamma, x : \tau_1, y : \tau'$ környezet egy permutációja, a 3.5. lemma alapján megkapjuk, hogy $\Gamma, y : \tau', x : \tau_1 \vdash e_2 : \tau_2$, így alkalmazhatjuk a 3.16. szabályt. \square

Helyettesítési lemma.

3.7. Lemma. *Ha $\Gamma, x : \tau \vdash e' : \tau'$ és $\Gamma \vdash e : \tau$, akkor $\Gamma \vdash e'[x \mapsto e] : \tau'$.*

Ez a lemma a modularitást fejezi ki: van két külön programunk, $e : \tau$ és $e' : \tau'$, utóbbi deklarálni egy τ típusú változót. e -t τ implementációjának, e' -t pedig e kliensének nevezzük. A 3.7. lemma azt mondja, hogy külön-külön típusellenőrizhetjük a két modult, és ekkor összetevés (linking) után is típushelyes lesz a programunk.

Bizonyítás. $\Gamma, x : \tau \vdash e' : \tau'$ levezetése szerinti indukció. A 3.10–3.11. szabályok esetén nem csinál semmit a helyettesítés. A 3.12–3.15. szabályok esetén az indukciós feltevésekből következik a helyettesítés helyessége. Pl. a 3.12. szabály esetén tudjuk, hogy $\Gamma, x : \tau \vdash e_1, e_2 : \text{int}$ és $\Gamma \vdash e_1[x \mapsto e], e_2[x \mapsto e] : \text{int}$, és azt szeretnénk belátni, hogy $\Gamma \vdash (e_1 + e_2)[x \mapsto e] : \text{int}$. Ez a helyettesítés definíciója alapján megegyezik azzal, hogy $\Gamma \vdash e_1[x \mapsto e] + e_2[x \mapsto e] : \text{int}$. Ezt a 3.12. szabály alkalmazásával megkapjuk. A 3.9 szabály alkalmazása esetén a kifejezésünk egy változó. Ha ez megegyezik x -szel, akkor a helyettesítés eredménye e lesz, és $\tau = \tau'$. Ekkor $\Gamma \vdash e : \tau$ miatt készen vagyunk. Ha a változónevek nem egyeznek meg, a helyettesítés nem csinál semmit. A 3.16. szabály esetén az indukciós feltevés alapján tudjuk, hogy $\Gamma \vdash e_1[x \mapsto e] : \tau_1$. Ezenkívül $\Gamma, x : \tau, y : \tau_1 \vdash e_2 : \tau_2$, ebből a 3.5. lemma alapján $\Gamma, y : \tau_1, x : \tau \vdash e_2 : \tau_2$,

majd az indukciós feltevés alapján kapjuk, hogy $\Gamma, y : \tau_1 \vdash e_2[x \mapsto e] : \tau_2$. Mivel $(\text{let } e_1 \text{ in } y.e_2)[x \mapsto e] = \text{let } e_1[x \mapsto e] \text{ in } y.e_2[x \mapsto e]$, a 3.16. szabály alapján kapjuk, hogy $\Gamma \vdash (\text{let } e_1 \text{ in } y.e_2)[x \mapsto e] : \tau_2$. \square

A helyettesítési lemma ellentéte a dekompozíció. Ez azt fejezi ki, hogy ha egy programrészlet többször előfordul egy programban, akkor azt kiemelhetjük (és ezzel rövidebbé, érthetőbbé, könnyebben karbantarthatóbbá tesszük a programot).

3.8. Lemma. *Ha $\Gamma \vdash e'[x \mapsto e] : \tau'$, akkor minden olyan τ -ra, melyre $\Gamma \vdash e : \tau$, $\Gamma, x : \tau \vdash e' : \tau'$.*

Bizonyítás. $\Gamma \vdash e'[x \mapsto e] : \tau'$ szerinti indukció. \square

Összefoglalás: a következő tulajdonságokat bizonyítottuk a szövegek és számok nyelvének típusrendszeréről.

- 3.2. lemma: nem triviális.
- 3.3. lemma: típusok unicitása.
- 3.4. lemma: típusozás inverziója.
- 3.5. lemma: környezet változóinak sorrendje nem számít.
- 3.6. lemma: gyengítési tulajdonság.
- 3.7. lemma: helyettesítés.
- 3.8. lemma: dekompozíció.

3.9. Feladat. *Típusozhatók -e az alábbi kifejezések az üres környezetben? Próbáljuk meg levezetni a típusukat.*

- $|"ab" \bullet "cd" + |\text{let } "e" \text{ in } x.x + x|$
- $|"ab" \bullet "cd" + |\text{let } "e" \text{ in } x.x \bullet x|$
- $|\text{let }|"ab" \bullet "cd" | \text{ in } x.x + x|$
- $|\text{let }|"ab" \bullet "cd" | \text{ in } x.x \bullet x|$
- $|(|"aaa"|)|$

3.10. Feladat. *Írjuk fel a 3.4. lemma összes esetét, és bizonyítsuk be őket.*

3.11. Feladat. *Bizonyítsuk be, hogy ha $\Gamma \vdash e : \tau$, akkor $\Gamma \text{ wf}$.*

3.12. Feladat. *Hogyan változna a típusrendszer, ha ugyanazt a $+$ szimbólumot használnánk szövegek összefűzésére és számok összeadására is? Változnának -e a típusrendszer tulajdonságai?*

További információ:

- A később tanulandó polimorf ill. függő típusrendszerekben nem igaz, hogy a környezet permutálható.
- A lineáris típusrendszerekben nem teljesül a gyengítési tulajdonság.

3.3. Operációs szemantika átíró rendszerrel

A szemantika a szintaxis jelentését adja meg. Ez megtehető valamilyen matematikai struktúrával, ilyenkor minden szintaktikus objektumhoz az adott struktúra valamely elemét rendeljük. Ezt denotációs szemantikának hívják, és nem tárgyaljuk. Az operációs szemantika azt írja le, hogy melyik kifejezéshez melyik másik kifejezést rendeljük, tehát a program hogyan fut.

Az operációs szemantika megadható átíró rendszerekkel.

Egy átíró rendszerben $e \mapsto e'$ alakú ítéleteket tudunk levezetni. Ez azt jelenti, hogy e kifejezés egy lépésben e' -re íródik át.

Az átírást iterálhatjuk, az iterált átíró rendszert az alábbi szabályokkal adjuk meg.

$$\overline{e \mapsto^* e} \quad (3.17)$$

$$\frac{e \mapsto e' \quad e' \mapsto^* e''}{e \mapsto^* e''} \quad (3.18)$$

A számok és szövegek nyelv bizonyos kifejezéseit *értékeknek* nevezzük. Értékek a zárt egész számok és a szövegek lesznek, melyekben a beágyazás operátorokon kívül más operátorok nincsenek. A program futását emiatt *kiértékelésnek* nevezzük: egy zárt kifejezésből értéket fogunk kapni.

Először megadjuk a nyelvünk értékeit az $e \text{ val}$ formájú ítélettel.

$$\overline{n \text{ val}} \quad (3.19)$$

$$\overline{''s'' \text{ val}} \quad (3.20)$$

Az átíró rendszer az alábbi szabályok adják meg. A rövidség kedvéért a bináris operátorokra vonatkozó szabályok egy részét összevontuk.

$$\frac{n_1 + n_2 = n}{n_1 + n_2 \mapsto n} \quad (3.21)$$

$$\frac{n_1 - n_2 = n}{n_1 - n_2 \mapsto n} \quad (3.22)$$

$$\frac{s_1 \text{ és } s_2 \text{ konkatenáltja } s}{''s_1'' \bullet ''s_2'' \mapsto ''s''} \quad (3.23)$$

$$\frac{s \text{ hossza } n}{|''s''| \mapsto n} \quad (3.24)$$

$$\frac{e_1 \mapsto e'_1}{e_1 \circ e_2 \mapsto e'_1 \circ e_2} \quad \circ \in \{+, -, \bullet\} \quad (3.25)$$

$$\frac{e_1 \text{ val} \quad e_2 \mapsto e'_2}{e_1 \circ e_2 \mapsto e_1 \circ e'_2} \quad \circ \in \{+, -, \bullet\} \quad (3.26)$$

$$\frac{e \mapsto e'}{|e| \mapsto |e'|} \quad (3.27)$$

$$\left[\frac{e_1 \mapsto e'_1}{\text{let } e_1 \text{ in } x.e_2 \mapsto \text{let } e'_1 \text{ in } x.e_2} \right] \quad (3.28)$$

$$\frac{[e_1 \text{ val}]}{\text{let } e_1 \text{ in } x.e_2 \mapsto e_2[x \mapsto e_1]} \quad (3.29)$$

A szemantikának két változata van: *érték szerinti* (by value) és *név szerinti* (by name) paraméterátadás. Utóbbinál elhagyjuk a zárójelezett 3.28. szabályt és a zárójelezett feltételt a 3.29. szabályból.

A 3.21–3.24. és a 3.29. szabályok utasítás szabályok, ezek adják meg, hogy ha egy operátornak már ki vannak értékelve a paraméterei, hogyan adjuk meg az eredményét. A 3.25–3.28. szabályok sorrendi szabályok, ezek adják meg, hogy milyen sorrendben történjék a kiértékelés. Például a $|"aa" + (3 - 2)|$ kifejezés kiértékelését kezddhetjük úgy, hogy először a szöveg hosszát értékeljük ki, majd a jobb oldali számot, de úgy is, hogy először a számot, majd a szöveg hosszát. A fenti operációs szemantika az előbbi fogja választani, minden operátornak először kiértékeljük az első paraméterét, majd a másodikat, majd alkalmazzuk az operátort az értékekre.

Példa kiértékelés:

$$\begin{array}{ll} & |"aa" + (3 - 2)| \\ 3.25 & \mapsto 2 + (3 - 2) \\ 3.22, 3.26 & \mapsto 2 + 1 \\ 3.21 & \mapsto 3 \end{array}$$

Érték szerinti paraméterátadásnál, mielőtt egy kifejezést hozzákötünk egy változóhoz, azt kiértékeljük. Így maximum egyszer értékelünk ki egy változót. Név szerinti paraméterátadás esetén nem értékeljük ki a kifejezést a kötés előtt, így ahányszor hivatkozunk rá, annyiszor fogjuk kiértékelni. Az érték szerinti paraméterátadás akkor pazarló, ha egyszer sem hivatkozunk a kötésre, a név szerinti akkor, ha több, mint egyszer hivatkozunk. A kettő előnyeit kombinálja az igény szerinti kiértékelés (call by need).

3.13. Feladat. *Értékeljük ki a $(1 + 1) - |"aa" \bullet "bb"|$ kifejezést, írjuk ki, hogy az egyes lépésekben melyik szabály(oka)t alkalmazzuk.*

3.14. Feladat. *Írjunk let kifejezést, amelynek kiértékelése megmutatja, hogy mi a különbség az érték szerinti és a név szerinti paraméterátadás között.*

3.15. Feladat. *Írjunk olyan let kifejezést, melynek kiértékelése érték szerinti paraméterátadásnál hatékonyabb.*

3.16. Feladat. *Írjunk olyan let kifejezést, melynek kiértékelése név szerinti paraméterátadásnál hatékonyabb.*

Nyílt kifejezések kiértékelése nem juttat el minket egy értékeléshez, egy adott ponton *elakad*.

3.17. Feladat. *Értékeljük ki az $x + (1 + 2)$ és az $(1 + 2) + x$ kifejezéseket!*

Ha szeretnénk valamit bizonyítani az átíró rendszerünkről, a szerkezeti indukciót alkalmazhatjuk rá. Ez azt jelenti, hogy ha $P(e \mapsto e')$ -t szeretnénk belátni minden $e \mapsto e'$ -re, akkor azt kell megmutatni, hogy a 3.21–3.29. szabályok megtartják P -t.

Például bebizonyítjuk az alábbi lemmát.

3.18. Lemma. *Nincs olyan e , hogy e val és $e \mapsto e'$ valamely e' -re.*

Bizonyítás. $e \mapsto e'$ szerinti indukció: egyik szabálykövetkezmény sem $n \mapsto e'$ vagy $"s" \mapsto e'$ alakú. \square

Determináltság.

3.19. Lemma. *Ha $e \mapsto e'$ és $e \mapsto e''$, akkor $e' = e''$.*

Bizonyítás. $e \mapsto e'$ és $e \mapsto e''$ szerinti indukció. \square

A nyelvünk különböző típusokhoz tartozó operátorai kétféle csoportba oszthatók: *bevezető- és eliminációs operátorokra*. *int* típusú kifejezések bevezető operátora a jelöletlen egész szám beágyazása operátor, eliminációs operátorai a $+$ és a $-$. A *sorrendi szabályok* azt mondják meg, hogy melyek egy eliminációs operátor *principális paraméterei* ($+$ és $-$ esetén mindkettő az), míg az utasítás szabályok azt adják meg, hogy ha a principális paraméterek a bevezető szabályokkal megadott alakúak, akkor hogyan kell kiértékelni az eliminációs operátort. Hasonlóképp, *str* típus esetén a $''-$ a bevezető operátor, míg $- \bullet -$ és $|-|$ az eliminációs operátorok. Az *utasítás szabályok* itt is hasonló szimmetriát mutat: megmondja, mit kapunk, ha az eliminációs operátorokat alkalmazzuk a bevezető operátorra. A változó bevezetése és a *let* szerkezeti operátorok, nem kapcsolódnak specifikus típusokhoz.

3.20. Feladat. *Mutassuk meg, hogy bármely s_1, s_2 -re létezik olyan e , hogy $''s_1'' \bullet ''s_2'' \mapsto^* e$ és $''s_1'' + ''s_2'' \mapsto^* e$.*

3.4. Típusrendszer és szemantika kapcsolata

A legtöbb programozási nyelv biztonságos, ami azt jelenti, hogy bizonyos hibák nem fordulhatnak elő a program futtatása során. Ezt úgy is nevezik, hogy a nyelv erős típusrendszerrel rendelkezik. A számok és szövegek nyelv esetén ez például azt jelenti, hogy nem fordulhat elő, hogy egy számhoz hozzáadunk egy szöveget, vagy két számot összefűzünk.

A típusmegőrzés (tárgyredukció, subject reduction, preservation) azt mondja ki, hogy ha egy típusozható kifejezésünk van, és egy átírási lépést végrehajtunk, ugyanazzal a típussal az átírt kifejezés is típusozható. $\cdot \vdash e : \tau$ helyett egyszerűen $e : \tau$ -t írunk.

3.21. Tétel. *Ha $e : \tau$ és $e \mapsto e'$, akkor $e' : \tau$.*

Bizonyítás. $e \mapsto e'$ szerinti indukció. A 3.21. szabály esetén tudjuk, hogy $n_1 + n_2 \mapsto n$, és $n_1 + n_2 : \tau$, és az inverziós lemmából (3.4) tudjuk, hogy $\tau = \text{int}$, és azt is tudjuk, hogy $n : \text{int}$. Hasonló a bizonyítás a 3.22–3.24. esetekben. A 3.25. esetén nézzük a $+$ esetet: tudjuk, hogy $e_1 + e_2 \mapsto e'_1 + e_2$, és, hogy $e_1 + e_2 : \tau$. Az inverziós lemma azt mondja, hogy $\tau = \text{int}$ és $e_1 : \text{int}$. Az indukciós hipotézisből azt kapjuk, hogy $e_1 \mapsto e'_1$ és $e'_1 : \text{int}$. Így az összeadás levezetési szabálya (3.12) megadja, hogy $e'_1 + e_2 : \text{int}$. A 3.26. esetén ugyanilyen az indoklás, csak a második paraméter változik. A 3.27. esetén tudjuk, hogy $|e| \mapsto |e'|$ és az indukciós hipotézisből és az inverzióból kapjuk, hogy $e' : \text{str}$, ebből a 3.15. szabály alapján kapjuk, hogy $|e'| : \text{int}$. A 3.28. szabály (érték szerinti paraméterátadás) esetén az inverzióból és az indukciós feltevésből tudjuk, hogy valamely τ_1 típusra $e'_1 : \tau_1$, és ebből a *let* típusozási szabálya (3.16) alapján kapjuk, hogy $\text{let } e'_1 \text{ in } x.e_2 : \tau_2$. A 3.29. szabály esetén inverzióból kapjuk, hogy $e_1 : \tau_1$ valamilyen τ_1 -re és $\cdot, x : \tau_1 \vdash \tau_2 : \tau_2$. Azt szeretnénk belátni, hogy $e_2[x \mapsto e_1] : \tau_2$. Ezt a helyettesítési lemmával (3.7) látjuk be. \square

Haladás (progress). Ez a lemma azt fejezi ki, hogy egy zárt, jól típusozott program nem akad el: vagy már ki van értékelve, vagy még egy átírási lépést végre tudunk hajtani.

3.22. Tétel. *Ha $e : \tau$, akkor vagy e val, vagy létezik olyan e' , hogy $e \mapsto e'$.*

A bizonyításhoz szükségünk van a következő lemmára a kanonikus alakokról.

3.23. Lemma. *Ha $e : \tau$ és e val, akkor ha*

- $\tau = \text{int}$, akkor $e = n$ valamely n -re,
- $\tau = \text{str}$, akkor $e = "s"$ valamely s -re.

Bizonyítás. e val és $e : \tau$ szerinti indukció. □

A tétel bizonyítása. $e : \tau$ levezetése szerinti indukció. A 3.9. levezetés nem fordulhat elő, mert a környezet üres. A 3.10–3.11. esetén már értékeink vannak, és nincs olyan átírási szabály, mely alkalmazható lenne. A 3.12. szabály esetén az indukciós feltevésből azt kapjuk, hogy $e_1 : \text{int}$ és vagy e_1 val vagy létezik $e_1 \mapsto e'_1$ lépés. Utóbbi esetben alkalmazhatjuk az $e_1 + e_2 \mapsto e'_1 + e_2$ lépést, előbbi esetben megnézzük a másik indukciós feltevést, mely e_2 -re vonatkozik. Ez azt mondja, hogy vagy e_2 val, vagy $e_2 \mapsto e'_2$. Utóbbi esetben alkalmazzuk az $e_1 + e_2 \mapsto e_1 + e'_2$ átírási szabályt, előbbi esetben tudjuk, hogy e_1 val és e_2 val, és hogy $e_1 : \text{int}$ és $e_2 : \text{int}$. A 3.23. lemmából és ebből kapjuk, hogy $e_1 = n_1$ és $e_2 = n_2$, így ha n_1 és n_2 összege n , akkor a 3.21. szabály szerinti $e_1 + e_2 \mapsto n$ átírást hajtjuk végre. A 3.13–3.14. esetben hasonló módon járunk el. A 3.15. esetben az indukciós feltevésből tudjuk, hogy $e : \text{str}$ és vagy e val vagy $e \mapsto e'$. Előbbi esetben a 3.23. lemmából kapjuk, hogy $e = "s"$ valamely s -re, és ha s hossza n , akkor végrehajtjuk a $|"s"| \mapsto n$ átírást (a 3.24. szabály), utóbbi esetben a 3.27. szabály alapján lépünk $|e| \mapsto |e'|$ -t. A 3.16. szabály használatakor név szerinti paraméterátadás esetén a $\text{let } e_1 \text{ in } x.e_2 \mapsto e_2[x \mapsto e_1]$ lépést tesszük meg (3.29), érték szerinti paraméterátadás esetén az indukciós feltevéstől függően tesszük meg a 3.29. vagy a 3.28. lépést. □

3.5. Futási idejű hibák

Ha a nyelvünkbe beteszünk egy $-/$ operátort, mely az egész osztást adja meg, a típusrendszert ill. az operációs szemantikát az alábbi szabályokkal egészítenénk ki.

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1/e_2 : \text{int}} \quad (3.30)$$

$$\frac{n_1 \text{ osztva } n_2\text{-vel } n}{n_1/n_2 \mapsto n} \quad (3.31)$$

A probléma, hogy az $n_1/0$ kifejezés kiértékelése elakad, nem igaz, hogy n_1 osztva 0-val n , bármilyen n -et is választunk. Ezt kétféleképpen lehet kezelni:

1. A típusrendszer kizárja az ilyen eseteket, tehát $n/0$ nem lesz típusozható. Ehhez az kell, hogy a típusrendszer eldöntse, hogy egy kifejezés 0 lesz-e, ha kiértékeljük. Ez nehéz.
2. Az ilyen típusú hibát futási időben ellenőrizzük, hibát ad a program futtatása.

Utóbbival foglalkozunk.

Az operációs szemantikát kiegészítjük az $e \text{ err}$ ítélettel, ami azt fejezi ki, hogy e kiértékelése hibával végződik. A következő levezetési szabályokkal egészítjük ki a 3.21–3.29. szabályokkal megadott átíró rendszert.

$$\frac{e_1 \text{ val}}{e_1 / 0 \text{ err}} \quad (3.32)$$

$$\frac{e_1 \text{ err}}{e_1 / e_2 \text{ err}} \quad (3.33)$$

$$\frac{e_1 \text{ val} \quad e_2 \text{ err}}{e_1 / e_2 \text{ err}} \quad (3.34)$$

Hasonlóan a többi szabály is propagálja a hibákat. Például az összeadás a következőképp.

$$\frac{e_1 \text{ err}}{e_1 + e_2 \text{ err}} \quad (3.35)$$

$$\frac{e_1 \text{ val} \quad e_2 \text{ err}}{e_1 + e_2 \text{ err}} \quad (3.36)$$

3.24. Feladat. *Add meg a többi operátorra is a hibák propagálási szabályait.*

Bevezetjük a hiba kifejezést. A nyelvet kiegészítjük egy **error** kifejezéssel.

$$\text{Exp} ::= \dots \mid \text{error} \quad (3.37)$$

A típusrendszert a következő szabállyal egészítjük ki.

$$\frac{\Gamma \text{ wf}}{\Gamma \vdash \text{error} : \tau} \quad (3.38)$$

Az operációs szemantika a következő szabállyal egészül ki.

$$\overline{\text{error err}} \quad (3.39)$$

3.25. Feladat. *Egészítsük ki a 3.22. tételt hibákkal. Tehát bizonyítsuk be, hogy ha $e : \tau$, akkor vagy $e \text{ err}$, vagy $e \text{ val}$, vagy létezik olyan e' , hogy $e \mapsto e'$.*

4. Függvények

4.1. Elsőrendű függvények

A számok és szövegek nyelvének típusait (3.1) lecseréljük a következőre.

$$\rho, \rho_1, \dots \in \text{BTy} ::= \text{str} \mid \text{int} \quad (4.1)$$

$$\tau, \tau', \dots \in \text{Ty} ::= \rho \mid \rho_1 \rightarrow \rho_2 \quad (4.2)$$

$\rho_1 \rightarrow \rho_2$ alakúak a függvény típusok, ahol ρ_1 és ρ_2 is alaptípus (base type). ρ_1 a függvény értelmezési tartománya, ρ_2 az értékkészlete.

4.1. Feladat. *Írjuk fel az összes lehetséges τ -t.*

A kifejezéseket kiegészítjük függvénnyel és függvény-alkalmazással.

$$e, e', \dots \in \text{Exp} ::= \dots \mid \text{fun}^\rho x.e \mid \text{app } e e' \quad (4.3)$$

A fun operátor aritása (BTy, Exp.Exp)Exp, második paraméterében köt egy Exp fajtájú kifejezést. Első paramétere (felső indexben) megadja a függvény érték-készletét.

A típusrendszert a következő szabályokkal egészítjük ki.

$$\frac{\Gamma, x : \rho_1 \vdash e_2 : \rho_2}{\Gamma \vdash \text{fun}^{\rho_1} x.e_2 : \rho_1 \rightarrow \rho_2} \quad (4.4)$$

$$\frac{\Gamma \vdash e : \rho_1 \rightarrow \rho_2 \quad \Gamma \vdash e_1 : \rho_1}{\Gamma \vdash \text{app } e e_1 : \rho_2} \quad (4.5)$$

4.2. Feladat. Írjuk fel azt az $\text{str} \rightarrow \text{str}$ típusú függvényt, ami egy szöveget háromszor egymás után másol.

Többparaméteres függvényt majd akkor tudunk megadni, ha bevezettük a szorzat típusokat (5. fejezet), ekkor pl. az összeadást elvégző függvény típusa $(\text{int} \times \text{int}) \rightarrow \text{int}$ lesz.

A 3.2–3.8. lemmák az elsőrendű függvényekkel kiegészített típusrendszerre is igazak, bizonyításuk ugyanolyan módon történik.

4.3. Feladat. A fentiek közül melyik lemma nem teljesülne, ha a fun operátor paraméterei közül kihagynánk a ρ alaptípust? Gondoljunk arra, hogy milyen típusa lehet ebben az esetben a $\text{fun } x.x$ kifejezésnek.

Az operációs szemantikát a következő szabályokkal egészítjük ki.

$$\overline{\text{fun}^\rho x.e \text{ val}} \quad (4.6)$$

$$\frac{e \mapsto e'}{\text{app } e e_1 \mapsto \text{app } e' e_1} \quad (4.7)$$

$$\left[\frac{e \text{ val} \quad e_1 \mapsto e'_1}{\text{app } e e_1 \mapsto \text{app } e e'_1} \right] \quad (4.8)$$

$$\frac{[e_1 \text{ val}]}{\text{app } (\text{fun}^\rho x.e_2) e_1 \mapsto e_2[x \mapsto e_1]} \quad (4.9)$$

A szögletes zárójelezett szabály ill. feltétel az érték szerinti paraméterátadás esetén szükséges.

A függvény típus bevezető operátora a fun, eliminációs operátora az app. A 4.7–4.8. szabályok sorrendi szabályok, míg a 4.9. szabály utasítás szabály, azt mondja meg, hogy mi történik, ha az eliminációs operátort a bevezető operátorra alkalmazzuk. Az app operátor principális paramétere az első paraméter.

Az így kiegészített operációs szemantikára is igaz, hogy nincs olyan kifejezés, mely egyszerre érték és át tudjuk írni (3.18. lemma) és determinisztikus (3.19. lemma).

Továbbá igaz a típusmegőrzés (3.21) és a haladás (3.22) tétele is, utóbbi bizonyításához a kanonikus alakok lemmáját (3.23) ki kell egészítenünk azzal, hogy ha $e : \rho_1 \rightarrow \rho_2$ és $e \text{ val}$, akkor $e = \text{fun}^{\rho_1} x.e'$ valamely x -re és e' -re, melyekre $x : \rho_1 \vdash e' : \rho_2$.

4.2. Magasabbrendű függvények

A típusrendszer egyszerűsödik, ha nem szorítjuk meg a függvénytípus értékkészletét és értelmezési tartományát alaptípusokra. A típusok a következők lesznek.

$$\tau, \tau', \dots \in \text{Ty} ::= \text{str} \mid \text{int} \mid \tau_1 \rightarrow \tau_2 \quad (4.10)$$

4.4. Feladat. Írjunk fel 10 különböző τ -t.

Kifejezések:

$$e, e', \dots \in \text{Exp} ::= \dots \mid \lambda^\tau x. e \mid e e' \quad (4.11)$$

A függvénydefiníciót most lambdával írjuk, az alkalmazást egyszerűen egymás mellé írással. A λ operátor aritása $(\text{Ty}, \text{Exp}, \text{Exp})\text{Exp}$ (az első paraméterét felső indexbe írjuk), az alkalmazásé $(\text{Exp}, \text{Exp})\text{Exp}$.

A típusrendszer csak abban különbözik az elsőrendű esettől, hogy tetszőleges τ típusokat engedélyezünk.

$$\frac{\Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \lambda^{\tau_1} x. e_2 : \tau_1 \rightarrow \tau_2} \quad (4.12)$$

$$\frac{\Gamma \vdash e : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_1 : \tau_1}{\Gamma \vdash e e_1 : \tau_2} \quad (4.13)$$

Most már tudunk többparaméteres függvényeket is megadni. Például egy $\text{int} \rightarrow (\text{str} \rightarrow \text{int})$ típusú függvény bemenete int , kimenete pedig egy újabb függvény, mely str -ből int -be képez. Úgy is tekinthetünk erre, mint egy két bemenettel, egy int és egy str típusúval rendelkező függvényre, melynek kimenete int . Ezt azzal is kihangsúlyozzuk, hogy a \rightarrow operátor jobbra zárójeleződik, tehát $\tau_1 \rightarrow \tau_2 \rightarrow \tau_3 = \tau_1 \rightarrow (\tau_2 \rightarrow \tau_3)$.

4.5. Feladat. Adjunk meg $\text{int} \rightarrow (\text{str} \rightarrow \text{int})$ típusú függvényt, mely kezd valamit az str típusú paraméterével is.

4.6. Feladat. Adjuk meg az $\text{apply}_3 : (\text{str} \rightarrow \text{str}) \rightarrow \text{str} \rightarrow \text{str}$ függvényt, mely az első paramétereként kapott függvényt háromszor alkalmazza a második paraméterében kapott szövegre.

A 3.2–3.8. lemmák a magasabbrendű függvényekkel kiegészített típusrendszerre is igazak. Pl. inverzió (3.4 lemma).

4.7. Lemma. *Tf. $\Gamma \vdash e : \tau$. Ekkor, ha $e = \lambda^{\tau_1} x. e_2$, akkor $\tau = \tau_1 \rightarrow \tau_2$ valamely τ_2 -re és $\Gamma, x : \tau_1 \vdash e_2 : \tau_2$. Ha $e = e' e_1$, akkor valamely τ_1 -re $\Gamma \vdash e' : \tau_1 \rightarrow \tau$ és $\Gamma \vdash e_1 : \tau_1$.*

Bizonyítás. $\Gamma \vdash e : \tau$ szerinti indukció. □

Az operációs szemantika az elsőrendű esettel analóg.

$$\overline{\lambda^\tau x. e \text{ val}} \quad (4.14)$$

$$\frac{e \mapsto e'}{\text{app } e e_1 \mapsto \text{app } e' e_1} \quad (4.15)$$

$$\left[\frac{e \text{ val} \quad e_1 \mapsto e'_1}{\text{app } e \ e_1 \mapsto \text{app } e \ e'_1} \right] \quad (4.16)$$

$$\frac{[e_1 \text{ val}]}{\text{app } (\lambda^\tau x. e_2) \ e_1 \mapsto e_2[x \mapsto e_1]} \quad (4.17)$$

A szögletes zárójelezett szabály ill. feltétel az érték szerinti paraméterátadás esetén szükséges.

Az így kiegészített operációs szemantikára is igaz, hogy nincs olyan kifejezés, mely egyszerre érték és át tudjuk írni (3.18. lemma) és determinisztikus (3.19. lemma).

Továbbá igaz a típusmegőrzés (3.21) és a haladás (3.22) tétele is, utóbbi bizonyításához a kanonikus alakok lemmáját (3.23) ki kell egészítenünk azzal, hogy ha $e : \tau_1 \rightarrow \tau_2$ és $e \text{ val}$, akkor $e = \lambda^{\tau_1} x. e'$ valamely x -re és e' -re, melyekre $x : \tau_1 \vdash e' : \tau_2$.

5. Véges adattípusok

Harper könyv: IV. rész.

5.1. Szorzat típusok

A bináris szorzat típusokkal rendezett párokat tudunk leírni. A projekciókkal ki tudjuk szedni a párban levő elemeket. A nulláris szorzat az egyelemű típus, mely nem hordoz információt, így nincsen eliminációs szabálya. Ezek általánosításai a véges szorzat típusok, melyeknek a felhasználó által megadott nevű projekciókkal rendelkező változatát nevezik rekordnak.

A szorzat típusok operációs szemantikája lehet *lusta* (lazy) és *mohó* (eager).² Lusta szemantika esetén egy tetszőleges (e, e') pár érték, míg mohó szemantika esetén szükséges, hogy e és e' már eleve értékek legyenek.

5.1.1. Nulláris és bináris szorzat típus

A szintaxis a következő.

$$\tau, \tau', \dots \in \text{Ty} \quad ::= \dots \mid \top \mid \tau \times \tau' \quad (5.1)$$

$$e, e', \dots \in \text{Exp} ::= \dots \mid \text{tt} \mid (e_1, e_2) \mid \text{proj}_1 e \mid \text{proj}_2 e \quad (5.2)$$

A nulláris szorzat típust egyelemű típusnak (top, unit) is nevezik, szokásos jelölései a \top -on kívül 1 és (). Az egyetlen elemét tt-vel (trivially true) jelöljük. A bináris szorzatot Descartes-szorzatnak is nevezik.

A típusrendszer a következő.

$$\frac{\Gamma \text{ wf}}{\Gamma \vdash \text{tt} : \top} \quad (5.3)$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2} \quad (5.4)$$

² A név szerinti és az érték szerinti paraméterátadás szemantikáról a függvény típusoknál beszélünk, véges adattípusoknál és induktív típusoknál (lásd később) lusta és mohó szemantikát mondunk.

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \text{proj}_1 e : \tau_1} \quad (5.5)$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \text{proj}_2 e : \tau_2} \quad (5.6)$$

5.1. Feladat. *Hány eleme van a $(\top \times \top) \times \top$ és a $\top \times (\top \times \top)$ típusoknak?*

Az operációs szemantika a következő.

$$\overline{\text{tt val}} \quad (5.7)$$

$$\frac{[e_1 \text{ val}] \quad [e_2 \text{ val}]}{(e_1, e_2) \text{ val}} \quad (5.8)$$

$$\left[\frac{e_1 \mapsto e'_1}{(e_1, e_2) \mapsto (e'_1, e_2)} \right] \quad (5.9)$$

$$\left[\frac{e_1 \text{ val} \quad e_2 \mapsto e'_2}{(e_1, e_2) \mapsto (e_1, e'_2)} \right] \quad (5.10)$$

$$\frac{e \mapsto e'}{\text{proj}_1 e \mapsto \text{proj}_1 e'} \quad (5.11)$$

$$\frac{e \mapsto e'}{\text{proj}_2 e \mapsto \text{proj}_2 e'} \quad (5.12)$$

$$\frac{[e_1 \text{ val}] \quad [e_2 \text{ val}]}{\text{proj}_1 (e_1, e_2) \mapsto e_1} \quad (5.13)$$

$$\frac{[e_1 \text{ val}] \quad [e_2 \text{ val}]}{\text{proj}_2 (e_1, e_2) \mapsto e_2} \quad (5.14)$$

A szögletes zárójelbe tett feltételek és szabályok csak a mohó szemantikában használatosak.

A 3.2–3.8, 3.18, 3.19 lemmák, és a 3.21) és a 3.22. tételek mind igazak erre a típusrendszerre is.

5.2. Feladat. *Bizonyítsd be a típusmegőrzés és a haladás tételét.*

5.1.2. Véges szorzat típusok

5.2. Összeg típusok

5.3. Izomorfizmusok