Nyelvek típusrendszere (jegyzet)

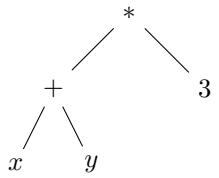
Kaposi Ambrus
Eötvös Loránd Tudományegyetem
akaposi@inf.elte.hu

2016. október 10.

1. Bevezető

Típusrendszerek helye a fordítóprogamokban:

1. Lexikális elemző (lexer): karakterek listájából tokenek listáját készíti, pl. `"(x + y) * 3"`-ból $\mathsf{brOpen}\,\mathsf{var}[x]\,\mathsf{op}[+]\,\mathsf{var}[y]\,\mathsf{brClose}\,\mathsf{op}[*]\,\mathsf{const}[3]$.
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 fákat, melyek leveleinél *változók* vannak, közbenső pontjaikon pedig *operárotok*. Például a természetes szám kifejezések és az ezekből és összeadásból álló kifejezések AST-it az alábbi definíciókkal adhatjuk meg.

$$n, n', ... \in \mathsf{Nat} ::= i \mid \mathsf{zero} \mid \mathsf{suc}\, n$$
$$e, e', ... \in \mathsf{Exp} ::= x \mid \mathsf{num}\, n \mid e + e'$$

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

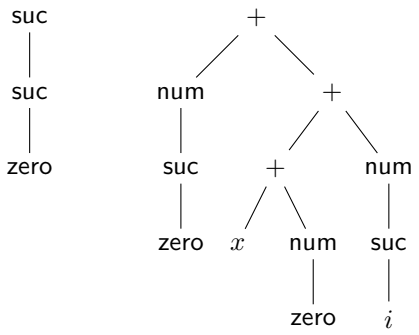
Az Exp fajtájú AST-ket e-vel és ennek vesszőzött változataival jelöljük, az Exp fajtájú változókat x-el jelöljük. Exp fajtájú AST-t egy unáris operátorral (num, aritása (Nat)Exp) és egy bináris operátorral (+, aritása (Exp, Exp)Exp) tudunk létrehozni. A num operátorral Nat fajtájú AST-ket tudunk kifejezésekbe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

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

$$i, i', i_1 ... \in \mathsf{Var}_{\mathsf{Nat}}$$
$$x, x', x_1, ... \in \mathsf{Var}_{\mathsf{Exp}}$$

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

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



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

Az olyan AST-ket, melyek nem tartalmaznak változót, *zártnak* nevezzük, egyébként *nyíltnak*. 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).

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

Néhány példa:

$$
\begin{array}{l}
(x + \mathsf{num}\,\mathsf{zero})[x \mapsto \mathsf{num}\,(\mathsf{suc}\,\mathsf{zero})] = \mathsf{num}\,(\mathsf{suc}\,\mathsf{zero}) + \mathsf{num}\,\mathsf{zero} \\
(x + x)[x \mapsto x' + \mathsf{num}\,\mathsf{zero}] = (x' + \mathsf{num}\,\mathsf{zero}) + (x' + \mathsf{num}\,\mathsf{zero}) \\
(x + \mathsf{num}\,(\mathsf{suc}\,i))[i \mapsto \mathsf{zero}] = x + \mathsf{num}\,(\mathsf{suc}\,\mathsf{zero})
\end{array}
$$

Megjegyezzük, hogy a $\mathsf{num}\,(\mathsf{suc}\,\mathsf{zero}) + \mathsf{num}\,(\mathsf{suc}\,\mathsf{zero})$ 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 $\mathsf{num}\,(\mathsf{suc}\,\mathsf{zero}) + \mathsf{num}\,(\mathsf{suc}\,\mathsf{zero}) = \mathsf{num}\,\big(\mathsf{suc}\,(\mathsf{suc}\,\mathsf{zero})\big)$, 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 $n \in \mathsf{Nat}$-ra szeretnénk belátni, hogy $\mathcal{P}(n)$, akkor elég azt belátni, hogy $\mathcal{P}(i)$ minden i-re, hogy $\mathcal{P}(\mathsf{zero})$ és hogy minden m-re $\mathcal{P}(m)$-ből következik $\mathcal{P}(\mathsf{suc}\,m)$. Ha minden $e \in \mathsf{Exp}$-re szeretnénk $\mathcal{Q}(e)$-t belátni, elég azt belátni, hogy minden x-re $\mathcal{Q}(x)$, hogy $n \in \mathsf{Nat}$-ra igaz $\mathcal{Q}(\mathsf{num}\,n)$ 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 *rekurziónak* hívjuk (structural recursion).

2.1. Feladat. *Végezzük el a következő helyettesítéseket:*

$$\big((x + x')[x \mapsto x']\big)[x' \mapsto x]$$
$$(x + x)[x \mapsto x' + x]$$
$$\big(\mathsf{suc}\,(\mathsf{suc}\,i)\big)[i \mapsto \mathsf{suc}\,i']$$
$$\big(\mathsf{num}\,(\mathsf{suc}\,i)\big)[x \mapsto \mathsf{num}\,\mathsf{zero}]$$

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

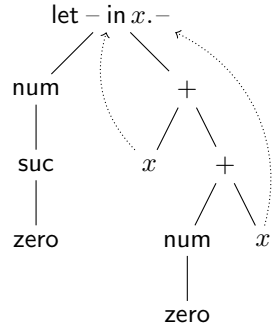
$$xs \in \mathsf{List}_{\mathsf{Nat}} ::= ?$$

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 $e \in \mathsf{Exp}_{x,x'}$ halmazban van. Ekkor pl. $e+x'' \in \in \mathsf{Exp}_{x,x',x''}$ és $e[x \mapsto \mathsf{num}\,\mathsf{zero}] \in \mathsf{Exp}_{x'}$. 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 monádjaiként (free monad) adjuk meg. Ezzel biztosítjuk, hogy pl. minden szintaxisfa véges.

2.2. Absztrakt kötéses fák

Az *absztrakt kötéses fák* (ABT, abstract binding tree) az AST-khez hasonlók, de változót *kötő* operátorok is szerepelhetnek benne. Ilyen például a $\mathsf{let}\,e\,\mathsf{in}\,x.e'$, 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 szintaxissal foglalkozunk, emiatt nem igaz, hogy $\mathsf{let}\,e\,\mathsf{in}\,x.x+x = e+e$). Azt mondjuk, hogy az x változó kötve van az e' kifejezésben. A let operátor aritását (Exp, Exp.Exp)Exp-el jelöljük, az operátor második paraméterében köt egy Exp fajtájú változót. Pl. $\mathsf{let}\,\mathsf{num}\,(\mathsf{suc}\,\mathsf{zero})\,\mathsf{in}\,x.x + (\mathsf{num}\,\mathsf{zero} + x)$ 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 $x + (\mathsf{num}\,\mathsf{zero} + x)$ 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.

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

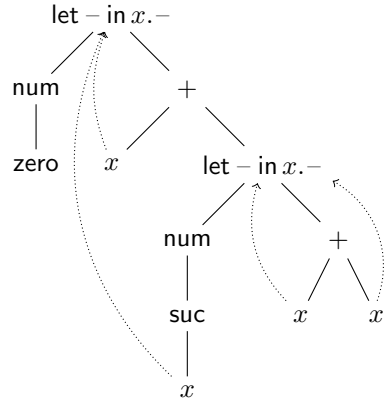
A $\mathsf{let\,num\,zero\,in}\, x.x + x$ ABT $x + x$ részfája tartalmaz egy *szabad változót*, az x-et, emiatt az $x + x$ ABT nyílt. Mivel $x + x$-ben csak az x a szabad változó, a kötés zárttá teszi a kifejezést, tehát a $\mathsf{let\,zero\,in}\, x.x + x$ ABT zárt. A kötések hatásköre a lehető legtovább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

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

$$\mathsf{let\,num\,zero\,in}\, x.\underline{x} + \underline{x}$$
$$\mathsf{let}\, \overline{y}\, \mathsf{in}\, x.\underline{x} + \underline{x}$$
$$\mathsf{let}\, \overline{y}\, \mathsf{in}\, x.\underline{x} + \mathsf{let}\, \underline{x}\, \mathsf{in}\, z.\underline{z}$$
$$\mathsf{let}\, \overline{y}\, \mathsf{in}\, x.\underline{x} + \mathsf{let}\, \overline{z}\, \mathsf{in}\, z.\underline{z}$$

A kötött változók csak pozíciókra mutatnak, a nevük nem érdekes. Például a $\mathsf{let\,num\,zero\,in}\, x.x + x$ és a $\mathsf{let\,num\,zero\,in}\, y.y + y$ 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. $\mathsf{let\,num\,zero\,in}\, x.x + \big(\mathsf{let\,num}\,(\mathsf{suc}\, x)\, \mathsf{in}\, x.x + x\big)$-ben az $x + x$-ben levő x-ek a második kötésre (ahol $\mathsf{num}\,(\mathsf{suc}\, x)$-et adtunk meg) mutat (a $\mathsf{num}\,(\mathsf{suc}\, x)$-ben levő x viszont az első kötésre mutat).



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

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

$$(\mathsf{let}\, x\, \mathsf{in}\, x'.x' + x'')[x'' \mapsto \mathsf{num\,zero}] = \mathsf{let}\, x\, \mathsf{in}\, x'.x' + \mathsf{num\,zero}$$
$$(\mathsf{let}\, x\, \mathsf{in}\, x'.x' + x'')[x' \mapsto \mathsf{num\,zero}] = \mathsf{let}\, x\, \mathsf{in}\, x'.x' + x''$$
$$(\mathsf{let}\, x\, \mathsf{in}\, x'.x' + x'')[x'' \mapsto x'] = \mathsf{let}\, x\, \mathsf{in}\, x'''.x''' + x'$$

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 azáltal, 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 köt[1].

$$(\mathsf{f}\,(y.t)\,t_1 \ldots t_n)[x \mapsto a] := \mathsf{f}\,\Big(\big(z.(t[y \mapsto z])\big)[x \mapsto a]\Big)\,(t_1[x \mapsto a]) \ldots (t_n[x \mapsto a])$$
$$\text{(ahol } z \text{ friss változónév)}$$

A biztonság kedvéért (lásd a fenti harmadik példát), a kötött változót átnevezzük 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 $n \in \mathsf{Nat}$-ra $\mathcal{Q}(n)$,
- minden $e, e' \in \mathsf{Exp}$-re, ha $\mathcal{Q}(e)$ és $\mathcal{Q}(e')$, akkor $\mathcal{Q}(e + e')$,
- minden $e, e' \in \mathsf{Exp}$-re és x változóra, ha $\mathcal{Q}(e)$ és $\mathcal{Q}(e')$, akkor $\mathcal{Q}(\mathsf{let}\,e\,\mathsf{in}\,x.e')$.

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

$$x + \mathsf{num}\,i$$
$$\mathsf{let}\,x\,\mathsf{in}\,x.\mathsf{num}\,i + x$$
$$\mathsf{let}\,x\,\mathsf{in}\,x.x' + \mathsf{let}\,x\,\mathsf{in}\,x'.x' + x$$
$$\mathsf{let}\,x\,\mathsf{in}\,x.x + \mathsf{let}\,x\,\mathsf{in}\,x.x + x$$
$$\mathsf{let}\,x\,\mathsf{in}\,x.(\mathsf{let}\,x\,\mathsf{in}\,x'.x'' + x') + \mathsf{let}\,x'\,\mathsf{in}\,x'.x' + x''$$

2.4. Feladat. *Lehet -e egy $i \in \mathsf{Var}_{\mathsf{Nat}}$ 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 $\mathsf{f}\,x.t\,t_1 \ldots t_n$ alakú és egy $\mathsf{f}\,x'.t'\,t'_1 \ldots t'_n$ alakú ABT egyenlő -e.*

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

$$(\mathsf{let}\,x\,\mathsf{in}\,x.x + x)[x \mapsto \mathsf{num}\,\mathsf{zero}]$$
$$(\mathsf{let}\,x'\,\mathsf{in}\,x.x' + x)[x' \mapsto \mathsf{num}\,\mathsf{zero}]$$
$$(\mathsf{let}\,x'\,\mathsf{in}\,x.x' + x')[x' \mapsto x]$$

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

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

$$x + \mathsf{num}\, i \stackrel{?}{=} x + \mathsf{num}\, i'$$

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

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

$$\mathsf{let}\, x\, \mathsf{in}\, x.x' + \mathsf{let}\, x\, \mathsf{in}\, x'.x' + x \stackrel{?}{=} \mathsf{let}\, x\, \mathsf{in}\, x'.x + \mathsf{let}\, x'\, \mathsf{in}\, x.x + x'$$

$$\mathsf{let}\, x\, \mathsf{in}\, x.x' + \mathsf{let}\, x\, \mathsf{in}\, x'.x' + x \stackrel{?}{=} \mathsf{let}\, x\, \mathsf{in}\, x''.x' + \mathsf{let}\, x''\, \mathsf{in}\, x.x + x''$$

$$\mathsf{let}\, x\, \mathsf{in}\, x.x + \mathsf{let}\, x\, \mathsf{in}\, x.x + x \stackrel{?}{=} \mathsf{let}\, x\, \mathsf{in}\, x'.x' + \mathsf{let}\, x'\, \mathsf{in}\, x'.x' + x'$$

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

$$a \in \mathsf{A} ::= y \mid \mathsf{d}\, y.a \mid \mathsf{g}\, a\, a$$
$$y, y', ... \in \mathsf{Var}_\mathsf{A}$$

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. $\mathsf{d}\, y.\mathsf{d}\, y'.y$ helyett $\mathsf{d}\,(\mathsf{d}\, 1)$-et írunk (0 mutatna a legközelebbi kötésre).

2.3. Levezetési fák

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

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

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

$$\frac{J_1 \qquad ... \qquad J_n}{J}$$

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

$$\frac{}{\mathsf{zero} + n' \text{ is } n'} \tag{2.1}$$

$$\frac{n + n' \text{ is } n''}{\mathsf{suc}\, n + n' \text{ is } \mathsf{suc}\, n''} \tag{2.2}$$

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 nincsen 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.

$$\dfrac{\dfrac{\dfrac{}{\mathsf{zero} + \mathsf{suc}\,\mathsf{zero}\ \mathsf{is}\ \mathsf{suc}\,\mathsf{zero}}\ (2.1)}{(\mathsf{suc}\,\mathsf{zero}) + \mathsf{suc}\,\mathsf{zero}\ \mathsf{is}\ \mathsf{suc}\,(\mathsf{suc}\,\mathsf{zero})}\ (2.2)}{\mathsf{suc}\,(\mathsf{suc}\,\mathsf{zero}) + \mathsf{suc}\,\mathsf{zero}\ \mathsf{is}\ \mathsf{suc}\,\big(\mathsf{suc}\,(\mathsf{suc}\,\mathsf{zero})\big)}\ (2.2)$$

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 nat ítélet azt fejezi ki, hogy n egy természetes szám. A következő szabályokkal tudjuk levezetni.

$$\frac{}{\mathsf{zero}\ \mathsf{nat}} \tag{2.3}$$

$$\frac{n\ \mathsf{nat}}{\mathsf{suc}\,n\ \mathsf{nat}} \tag{2.4}$$

Ezek azt fejezik ki, hogy a 0 természetes szám, és bármely természetes szám rákövetkezője is az. N.b. azt nem tudjuk levezetni, hogy i 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 $\mathsf{isBalanced}$ n.

$$\frac{}{x\ \mathsf{isBalanced}\ \mathsf{zero}} \tag{2.5}$$

$$\frac{}{\mathsf{num}\,n\ \mathsf{isBalanced}\ \mathsf{zero}} \tag{2.6}$$

$$\frac{e\ \mathsf{isBalanced}\ n \qquad e'\ \mathsf{isBalanced}\ n}{e + e'\ \mathsf{isBalanced}\ \mathsf{suc}\,n} \tag{2.7}$$

Egy példa levezetés.

$$\dfrac{\dfrac{\dfrac{}{x'\ \mathsf{isBalanced}\ \mathsf{zero}} \quad \dfrac{}{x\ \mathsf{isBalanced}\ \mathsf{zero}}}{x + x\ \mathsf{isBalanced}\ \mathsf{suc}\,\mathsf{zero}} \quad \dfrac{\dfrac{}{x'\ \mathsf{isBalanced}\ \mathsf{zero}} \quad \dfrac{}{\mathsf{num}\,\mathsf{zero}\ \mathsf{isBalanced}\ \mathsf{zero}}}{x' + \mathsf{num}\,i\ \mathsf{isBalanced}\ \mathsf{suc}\,\mathsf{zero}}}{(x + x) + (x' + \mathsf{num}\,i)\ \mathsf{isBalanced}\ \mathsf{suc}\,(\mathsf{suc}\,\mathsf{zero})}$$

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 $J_1, ..., J_n$ és következménye J, ha $\mathcal{P}(J_1), ..., \mathcal{P}(J_n)$ mind teljesül, akkor $\mathcal{P}(J)$ is.

A szerkezeti indukció konkrét használatára mutatunk két példát.

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

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

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

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

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

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

- Ha a 2.5 szabályt használtuk $e\ \mathsf{isBalanced}\ n$ levezetésére, akkor azt kell belátnunk, hogy $x[x_1 \mapsto e_1]\ \mathsf{isBalanced}\ \mathsf{zero}$. Ha $x = x_1$, akkor ez azzal egyezik meg, hogy $e_1'\ \mathsf{isBalanced}\ \mathsf{zero}$, 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 $(\mathsf{num}\, n)[x_1 \mapsto e_1]\ \mathsf{isBalanced}\ \mathsf{zero}$, 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 $e[x_1 \mapsto e_1]\ \mathsf{isBalanced}\ n$ és $e'[x_1 \mapsto e_1]\ \mathsf{isBalanced}\ n$, és azt szeretnénk belátni, hogy $(e + e')[x_1 \mapsto e_1]\ \mathsf{isBalanced}\ \mathsf{suc}\, n$, de a helyettesítés definíciója alapján ez megegyezik $e[x_1 \mapsto e_1] + e'[x_1 \mapsto e_1]\ \mathsf{isBalanced}\ \mathsf{suc}\, n$-al, amit pedig a 2.7 szabály alapján látunk.

□

2.11. Feladat. *Adjuk meg a* $\mathsf{max}\, n\, n' = n''$ *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* $\mathsf{e}\ \mathsf{hasHeight}\ \mathsf{n}$ *ítélet levezetési szabályait.*

2.13. Feladat. *Adjuk meg a* $\mathsf{isEven}\, n$ *és* $\mathsf{isOdd}\, n$ *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-kkel.

3.1. Szintaxis

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

$\tau, \tau', ... \in \mathsf{Ty}$	$::= \mathsf{int}$	egész számok típusa
	$\mid \mathsf{str}$	szövegek típusa
$e, e', ... \in \mathsf{Exp}$	$::= x$	változó
	$\mid n$	egész szám beágyazása
	$\mid$ "s"	szöveg beágyazása
	$\mid e + e'$	összeadás
	$\mid e - e'$	kivonás
	$\mid e \bullet e'$	összefűzés
	$\mid \lvert e \rvert$	hossz
	$\mid \mathsf{let}\, e\, \mathsf{in}\, x.e'$	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`" $\bullet\, x \bullet$ "`b`" $\bullet\, x$.

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

3.1. Feladat. *Írjuk fel az összes operátor aritását!*

3.2. Típusrendszer

A típusrendszer megszorítja a leírható kifejezéseket azzal a céllal, hogy az értelmetlen kifejezéseket kiszűrje. Pl. a $\lvert 3 \rvert$ 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.

$$\Gamma, \Gamma', ... \in \mathsf{Con} ::= \cdot \mid \Gamma, x : \tau$$

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.

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

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

$$\frac{}{\cdot\,\mathsf{wf}} \tag{3.1}$$

$$\frac{\Gamma\,\mathsf{wf} \qquad x \notin \mathsf{dom}(\Gamma)}{\Gamma, x : \tau\,\mathsf{wf}} \tag{3.2}$$

Ezután csak jól formált környezetekkel fogunk dolgozni.

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

$$\frac{\Gamma\,\mathsf{wf} \qquad x \notin \mathsf{dom}(\Gamma)}{(x : \tau) \in \Gamma, x : \tau} \tag{3.3}$$

$$\frac{(x : \tau) \in \Gamma \qquad y \notin \mathsf{dom}(\Gamma)}{(x : \tau) \in \Gamma, y : \tau'} \tag{3.4}$$

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.

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x : \tau} \tag{3.5}$$

$$\frac{\Gamma\,\mathsf{wf}}{\Gamma \vdash n : \mathsf{int}} \tag{3.6}$$

$$\frac{\Gamma\,\mathsf{wf}}{\Gamma \vdash \text{"}s\text{"} : \mathsf{str}} \tag{3.7}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{int} \qquad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1 + e_2 : \mathsf{int}} \tag{3.8}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{int} \qquad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1 - e_2 : \mathsf{int}} \tag{3.9}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{str} \qquad \Gamma \vdash e_2 : \mathsf{str}}{\Gamma \vdash e_1 \bullet e_2 : \mathsf{str}} \tag{3.10}$$

$$\frac{\Gamma \vdash e : \mathsf{str}}{\Gamma \vdash |e| : \mathsf{int}} \tag{3.11}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau_2 \qquad x \notin \mathsf{dom}(\Gamma)}{\Gamma \vdash \mathsf{let}\, e_1 \,\mathsf{in}\, x.e_2 : \tau_2} \tag{3.12}$$

Példa levezetés:

$$\dfrac{\dfrac{\dfrac{}{\cdot\,\mathsf{wf}}\,3.1}{\cdot \vdash \text{"}\mathsf{a}\text{"} : \mathsf{str}}\,3.7 \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot\,\mathsf{wf}}\,3.1 \qquad x \notin \{\}}{(x : \mathsf{str}) \in \cdot, x : \mathsf{str}}\,3.3}{\cdot, x : \mathsf{str} \vdash x : \mathsf{str}}\,3.5 \qquad \dfrac{\dfrac{}{\cdot\,\mathsf{wf}}\,3.1 \qquad x \notin \{\}}{\cdot, x : \mathsf{str}\,\mathsf{wf}}\,3.2}{\cdot, x : \mathsf{str} \vdash |x| : \mathsf{int}}\,3.11 \qquad \dfrac{}{\cdot, x : \mathsf{str} \vdash 2 : \mathsf{int}}\,3.6}{\cdot, x : \mathsf{str} \vdash |x| + 2 : \mathsf{int}}\,3.8}{\cdot \vdash \mathsf{let}\,\text{"}\mathsf{a}\text{"}\,\mathsf{in}\, x.|x| + 2 : \mathsf{int}}\,3.12$$

A $|3|$ kifejezés nem típusozható, mert csak a 3.11. 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 esetnél 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* $\tau = \mathsf{int}$*,* $\Gamma \vdash e_1 : \mathsf{int}$ *és* $\Gamma \vdash e_2 : \mathsf{int}$*. 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 3.6–3.11. szabályoknál csak az induktív hipotéziseket, majd a szabályt használjuk. A 3.12. szabály esetén azt szeretnénk belátni, hogy $\Gamma' \vdash \mathsf{let}\, e_1 \,\mathsf{in}\, x.e_2 : \tau_2$, feltéve, hogy $\Gamma' \vdash$ $\vdash e_1 : \tau_1$ 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 3.12. szabály alapján megkapjuk, hogy $\Gamma' \vdash \mathsf{let}\, e_1 \,\mathsf{in}\, x.e_2 : \tau_2$. A 3.5. 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 3.2 szabályt, ahányadik komponens $(x : \tau)$ hátulról Γ'-ben. □

A típusrendszer egy további fontos tulajdonsága a gyengítési tulajdonság.

3.6. Lemma. *Ha* $\Gamma \vdash e : \tau$ *levezethető, és* $y \notin \mathsf{dom}(\Gamma)$*, 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 $P(\Gamma \vdash e : \tau) = \Gamma, y : \tau' \vdash$ $\vdash e : \tau$, ahol $y \notin \mathsf{dom}(\Gamma)$. A változó esetében eggyel többször használjuk a 3.4. szabályt, a 3.6–3.7. szabályok esetén ugyanazeket a szabályokat alkalmazzuk más környezetekre, a 3.8–3.11. szabályok esetén az indukciós feltevést használjuk, majd magát a szabályt. A 3.12. 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 $\Gamma, y : \tau' \vdash \mathsf{let}\, e_1 \,\mathsf{in}\, x.e_2 : \tau_2$. 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 3.12. szabályt. □

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 deklarál 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 3.6–3.7. szabályok esetén nem csinál semmit a helyettesítés. A 3.8–3.11. szabályok esetén az indukciós feltevésekből következik a helyettesítés helyessége. Pl. a 3.8. szabály esetén tudjuk, hogy $\Gamma, x : \tau \vdash e_1, e_2 : \mathsf{int}$ és $\Gamma \vdash e_1[x \mapsto e], e_2[x \mapsto e] : \mathsf{int}$, és azt szeretnénk belátni, hogy $\Gamma \vdash (e_1 + e_2)[x \mapsto e] : \mathsf{int}$. Ez a helyettesítés definíciója alapján megegyezik azzal, hogy $\Gamma \vdash e_1[x \mapsto e] + e_2[x \mapsto e] : \mathsf{int}$. Ezt a 3.8. szabály alkalmazásával megkapjuk. A 3.5 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 3.12. 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 $(\mathsf{let}\, e_1\, \mathsf{in}\, y.e_2)[x \mapsto e] = \mathsf{let}\, e_1[x \mapsto e]\, \mathsf{in}\, y.e_2[x \mapsto e]$, a 3.12. szabály alapján kapjuk, hogy $\Gamma \vdash (\mathsf{let}\, e_1\, \mathsf{in}\, y.e_2)[x \mapsto e] : \tau_2$. □

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. □

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.*

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

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* $\Gamma\,\mathsf{wf}$*.*

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 $e \longmapsto e'$ 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.

$$\frac{}{e \longmapsto^* e} \tag{3.13}$$

$$\frac{e \longmapsto e' \qquad e' \longmapsto^* e''}{e \longmapsto^* e''} \tag{3.14}$$

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 ki*értékelés*nek nevezzük: egy zárt kifejezésből értéket fogunk kapni.

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

$$\frac{}{n\,\mathsf{val}} \tag{3.15}$$

$$\frac{}{\text{"}s\text{"}\,\mathsf{val}} \tag{3.16}$$

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

$$\frac{n_1 + n_2 = n}{n_1 + n_2 \longmapsto n} \tag{3.17}$$

$$\frac{n_1 - n_2 = n}{n_1 - n_2 \longmapsto n} \tag{3.18}$$

$$\frac{s_1 \text{ és } s_2 \text{ konkatenáltja } s}{\text{"}s_1\text{"} \bullet \text{"}s_2\text{"} \longmapsto \text{"}s\text{"}} \tag{3.19}$$

$$\frac{s \text{ hossza } n}{|\text{"}s\text{"}| \longmapsto n} \tag{3.20}$$

$$\frac{e_1 \mapsto e_1'}{e_1 \circ e_2 \mapsto e_1' \circ e_2} \circ \in \{+, -, \bullet\} \tag{3.21}$$

$$\frac{e_1\ \mathsf{val} \qquad e_2 \mapsto e_2'}{e_1 \circ e_2 \mapsto e_1 \circ e_2'} \circ \in \{+, -, \bullet\} \tag{3.22}$$

$$\frac{e \longmapsto e'}{|e| \longmapsto |e'|} \tag{3.23}$$

$$\left[\frac{e_1 \mapsto e_1'}{\mathsf{let}\ e_1\ \mathsf{in}\ x.e_2 \longmapsto \mathsf{let}\ e_1'\ \mathsf{in}\ x.e_2}\right] \tag{3.24}$$

$$\frac{[e_1\ \mathsf{val}]}{\mathsf{let}\ e_1\ \mathsf{in}\ x.e_2 \longmapsto e_2[x \mapsto e_1]} \tag{3.25}$$

A szemantikának két változata van: *érték szerinti* (by value) és *név szerinti* (by name) paraméterátadás. Utóbbinál elhagyjuk a zárójelezett 3.24. szabályt és a zárójelezett feltételt a 3.25. szabályból.

A 3.17–3.20. és a 3.25. szabályok utasítás szabályok, ezek adják meg, hogy ha egy operátornak már ki vannak értékelve a paraméterei, hogyan adjuk meg az eredményét. A 3.21–3.24. szabályok sorrendi szabályok, ezek adják meg, hogy milyen sorrendben történjék a kiértékelés. Például a $|\text{"}\mathtt{aa}\text{"}| + (3 - 2)$ kifejezés kiértékelését kezdhetjük 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 előbbit fogja választani, minden operátornak először kiértékeljük az első paraméterét, majd a másodikat, majd alkalmazzuk az operátort az értékekre.

Példa kiértékelés:

$$\begin{aligned} & |\text{"}\mathtt{aa}\text{"}| + (3 - 2) \\ 3.21 \quad & \longmapsto 2 + (3 - 2) \\ 3.18, 3.22 \quad & \longmapsto 2 + 1 \\ 3.17 \quad & \longmapsto 3 \end{aligned}$$

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 a lusta kiértékelés (call by need), a 3.5. fejezetben tárgyaljuk.

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

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 $P(e \longmapsto e')$-t szeretnénk belátni minden $e \longmapsto e'$-re, akkor azt kell megmutatni, hogy a 3.17–3.25. szabályok megtartják P-t.

Például bebizonyítjuk az alábbi lemmát.

3.18. Lemma. *Nincs olyan* e*, hogy* $e\,\mathsf{val}$ *és* $e \longmapsto e'$ *valamely* e'*-re.*

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

Determináltság.

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

Bizonyítás. $e \longmapsto e'$ és $e \longmapsto e''$ szerinti indukció. □

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 − 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* $|"s_1" \bullet "s_2"| \longmapsto^* e$ *és* $|"s_1"| + |"s_2"| \longmapsto^* e$*.*

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* $e \longmapsto e'$*, akkor* $e' : \tau$*.*

Bizonyítás. $e \longmapsto e'$ szerinti indukció. A 3.17. szabály esetén tudjuk, hogy $n_1 + +n_2 \longmapsto n$, és $n_1+n_2 : \tau$, és az inverziós lemmából (3.4) tudjuk, hogy $\tau = \mathsf{int}$, és azt is tudjuk, hogy $n : \mathsf{int}$. Hasonló a bizonyítás a 3.18–3.20. esetekben. A 3.21. esetén nézzük a + esetet: tudjuk, hogy $e_1+e_2 \longmapsto e_1'+e_2$, és, hogy $e_1+e_2 : \tau$. Az inverziós lemma azt mondja, hogy $\tau = \mathsf{int}$ és $e_1 : \mathsf{int}$. Az indukciós hipotézisből azt kapjuk, hogy $e_1 \longmapsto e_1'$ és $e_1' : \mathsf{int}$. Így az összeadás levezetési szabálya (3.8) megadja, hogy $e_1'+e_2 : \mathsf{int}$. A 3.22. esetén ugyanilyen az indoklás, csak a második paraméter változik. A 3.23. esetén tudjuk, hogy $|e| \longmapsto |e'|$ és az indukciós hipotézisből és az inverzióból kapjuk, hogy $e' : \mathsf{str}$, ebből a 3.11. szabály alapján kapjuk, hogy $|e'| : \mathsf{int}$. A 3.24. 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 (3.12) alapján kapjuk, hogy $\mathsf{let}\, e_1' \,\mathsf{in}\, x.e_2 : \tau_2$. A 3.25. 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. □

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 $e\,\mathsf{val}$, vagy létezik olyan e', hogy $e' : \tau$.*

A bizonyításhoz szükségünk van a következő lemmára a kanonikus alakokról.

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

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

Bizonyítás. $e\,\mathsf{val}$ és $e : \tau$ szerinti indukció. □

A tétel bizonyítása. $e : \tau$ levezetése szerinti indukció. A 3.5. levezetés nem fordulhat elő, mert a környezet üres. A 3.6–3.7. esetén már értékeink vannak, és nincs olyan átírási szabály, mely alkalmazható lenne. A 3.8. szabály esetén az indukciós feltevésből azt kapjuk, hogy $e_1 : \mathsf{int}$ és vagy $e_1\,\mathsf{val}$ vagy létezik $e_1 \longmapsto e_1'$ lépés. Utóbbi esetben alkalmazhatjuk az $e_1+e_2 \longmapsto e_1'+e_2$ lépést, előbbi esetben megnézzük a másik indukciós feltevést, mely e_2-re vonatkozik. Ez azt mondja, hogy vagy $e_2\,\mathsf{val}$, vagy $e_2 \mapsto e_2'$. Utóbbi esetben alkalmazzuk az $e_1 + e_2 \longmapsto e_1 + +e_2'$ átírási szabályt, előbbi esetben tudjuk, hogy $e_1\,\mathsf{val}$ és $e_2\,\mathsf{val}$, és hogy $e_1 : \mathsf{int}$ és $e_2 : \mathsf{int}$. 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 3.17. szabály szerinti $e_1 + e_2 \longmapsto n$ átírást hajtjuk végre. A 3.9–3.10. esetben hasonló módon járunk el. A 3.11. esetben az indukciós feltevésből tudjuk, hogy $e : \mathsf{str}$ és vagy $e\,\mathsf{val}$ vagy $e \longmapsto e'$. Előbbi esetben a 3.23. lemmából kapjuk, hogy $e =$ "s" valamely s-re, és ha s hossza n, akkor végrehajtjuk a $|$"s"$| \longmapsto n$ átírást (a 3.20. szabály), utóbbi esetben a 3.23. szabály alapján lépünk $|e| \longmapsto |e'|$-t. A 3.12. szabály használatakor név szerinti paraméterátadás esetén a $\mathsf{let}\, e_1 \,\mathsf{in}\, x.e_2 \longmapsto e_2[x \mapsto e_1]$ lépést tesszük meg (3.25), érték szerinti paraméterátadás esetén az indukciós feltevéstől függően tesszük meg a 3.25. vagy a 3.24. lépést. □

3.5. Nagy lépés szemantika

Big step semantics.
Lusta kiértékelés.

4. Függvények

5. Véges adattípusok

6. Végtelen adattípusok

7. Parcialitás, rekurzív típusok

8. Imperatív programozás

9. Polimorfizmus

10. Altípus

11. Függő típus

12. Megoldások