

Nyelvek típusrendszere (jegyzet)

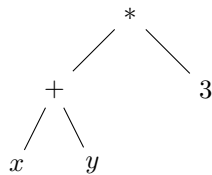
Kaposi Ambrus
Eötvös Loránd Tudományegyetem
akaposi@inf.elte.hu

2016. szeptember 27.

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

2.1. Absztrakt szintaxisfák

Absztrakt szintaxisfának (AST, abstract syntax tree) nevezzük az olyan fákat, melyek leveleinél *változók* vannak, közben 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 fajtajú AST-eket n -el, n' -vel stb. jelöljük. Nat fajtajú 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 fajtajú AST-t jelöl, míg i maga egy Nat fajtajú AST, mely egy darab változóból áll.

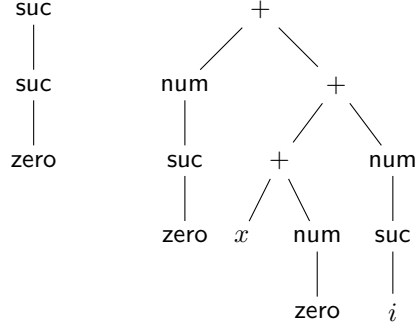
Az Exp fajtajú AST-eket e -vel és ennek vesszőzött változataival jelöljük, az Exp fajtajú változókat x -el jelöljük. Exp fajtajú AST-t egy unáris operátorral (num, *aritása* $(\text{Nat})\text{Exp}$) és egy bináris operátorral (+, *aritása* $(\text{Exp}, \text{Exp})\text{Exp}$) tudunk létrehozni. A num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

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

Megjegyezzük, hogy a $\text{num}(\text{suc zero}) + \text{num}(\text{suc zero})$ 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 $\text{num}(\text{suc zero}) + \text{num}(\text{suc zero}) = \text{num}(\text{suc}(\text{suc zero}))$, 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 *strukturális indukciónak* nevezzük.

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 *strukturális rekúrzió*-nak hívjuk.

2.1. Feladat. Végezzük el a következő helyettesítéseket:

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

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

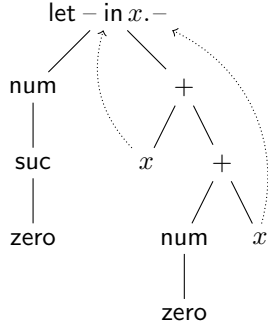
$$xs \in \text{List}_{\text{Nat}} ::= ?$$

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 monádjaiként (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. $\text{let num}(\text{succ zero}) \text{ in } x.x + (\text{num zero} + x)$ 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 $x + x$ részfaja 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.

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

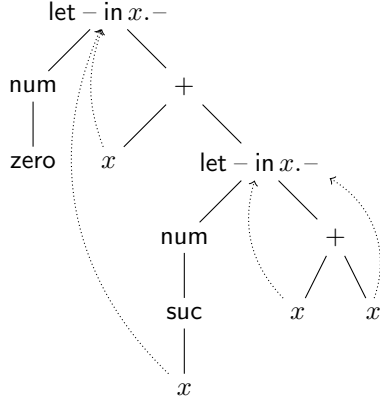
$\text{let } \bar{y} \text{ in } x.\underline{x} + \underline{x}$

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

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

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 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 köt¹.

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

(ahol z 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 strukturális 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 $\mathcal{Q}(n)$,
- 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 fájtájú 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]

```

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!*

$$\begin{aligned}
x + \text{num } i &\stackrel{?}{=} x + \text{num } i' \\
\text{let } x \text{ in } x.\text{num } i + x &\stackrel{?}{=} \text{let } x \text{ in } x'.\text{num } i + x' \\
\text{let } x \text{ in } x.\text{num } i + x &\stackrel{?}{=} \text{let } x \text{ in } x.\text{num } i + x' \\
\text{let } x \text{ in } x.x' + \text{let } x \text{ in } x'.x' + x &\stackrel{?}{=} \text{let } x \text{ in } x'.x + \text{let } x' \text{ in } x.x + x' \\
\text{let } x \text{ in } x.x' + \text{let } x \text{ in } x'.x' + x &\stackrel{?}{=} \text{let } x \text{ in } x''.x' + \text{let } x'' \text{ in } x.x + x'' \\
\text{let } x \text{ in } x.x + \text{let } x \text{ in } x.x + x &\stackrel{?}{=} \text{let } x \text{ in } x'.x' + \text{let } x' \text{ in } x'.x' + x'
\end{aligned}$$

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

$$\begin{aligned}
a \in A &::= y \mid d \ y.a \mid g \ a \ a \\
y, y', \dots &\in \text{Var}_A
\end{aligned}$$

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.

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

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

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 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.

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

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.

$$\frac{}{\text{zero nat}}(2.3)$$

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

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 isBalanced n .

$$\frac{}{x \text{ isBalanced zero}}(2.5)$$

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

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

Egy példa levezetés.

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

Ha valamit be szeretnénk bizonyítani minden levezethető ítéletről, ehhez a szabályok szerinti strukturális 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 strukturális 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. A természetes számok levezetése alapján szabály indukció. A fenti P -t úgy választjuk meg, hogy $P(n \text{ nat}) := n + \text{zero}$ is n . Ha a 2.3 szabályt használtuk, akkor $P(\text{zero nat}) = \text{zero} + \text{zero}$ is zero -t kell bizonyítanunk, ezt megteesszük a 2.1 szabállyal. Ha a 2.4 szabályt használtuk, akkor az indukciós hipotézis azt mondja, hogy $P(n \text{ nat}) = n + \text{zero}$ is n , nekünk pedig azt kell bizonyítani, hogy $P(\text{suc } n \text{ nat}) = \text{suc } n + \text{zero}$ is $\text{suc } n$. 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 isBalanced zero és e isBalanced n , akkor bármely x_1 -re $e[x_1 \mapsto e_1]$ isBalanced n .*

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

- Ha a 2.5 szabályt használtuk e isBalanced n levezetésére, akkor azt kell belátnunk, hogy $e[x_1 \mapsto e_1]$ isBalanced zero. Ha $x = x_1$, akkor ez azzal egyezik meg, hogy e'_1 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 $\text{suc } n$, de a helyettesítés definíciója alapján ez megegyezik $e[x_1 \mapsto e_1] + e'[x_1 \mapsto e_1]$ isBalanced $\text{suc } n$ -al, amit pedig a 2.7 szabály alapján látunk.

\square

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 $e \text{ hasHeight } n$ 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 strukturális indukció elvét ebben az esetben az eliminátor fejezi ki.

3. Számok és szövegek

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.

$\tau, \tau', \dots \in \text{Ty} ::= \text{int}$	egész számok típusa
$\quad \quad \quad \text{str}$	szövegek típusa
$e, e', \dots \in \text{Exp} ::= x$	változó
$\quad \quad \quad n$	egész szám beágyazása
$\quad \quad \quad "s"$	szöveg beágyazása
$\quad \quad \quad e + e'$	összeadás
$\quad \quad \quad e - e'$	kivonás
$\quad \quad \quad e \bullet e'$	összefűzés
$\quad \quad \quad e $	hossz
$\quad \quad \quad \text{let } e \text{ in } x.e'$	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" • 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 megszorítja 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 típushibákat szűr ki a típusrendszer, az nincs 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.

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

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.

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

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

$$\overline{\cdot \text{ wf}} \tag{3.1}$$

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

Ezután csak jól formált környezetekkel fogunk dolgozni.

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

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

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

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.

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

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

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

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}} \tag{3.8}$$

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 - e_2 : \text{int}} \tag{3.9}$$

$$\frac{\Gamma \vdash e_1 : \mathbf{str} \quad \Gamma \vdash e_2 : \mathbf{str}}{\Gamma \vdash e_1 \bullet e_2 : \mathbf{str}} \quad (3.10)$$

$$\frac{\Gamma \vdash e : \mathbf{str}}{\Gamma \vdash |e| : \mathbf{int}} \quad (3.11)$$

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

Példa levezetés:

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.

Egyrészt a típusrendszer nem triviális, tehát nem igaz, hogy minden kifejezés típusozható (pl. az előbbi [3]).

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

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).

Bizonyítás. A levezetés szerinti indukcióval.

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

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 $\Gamma' \vdash \text{let } e_1 \text{ in } x.e_2 : \tau_2$, 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 3.12. szabály alapján megkapjuk, hogy $\Gamma' \vdash \text{let } e_1 \text{ in } x.e_2 : \tau_2$. 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. \square

A típusrendszer egy további fontos tulajdonsága a gyengítési tulajdonság.

3.5. 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 3.4. 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 $\Gamma, y : \tau' \vdash \text{let } e_1 \text{ in } x.e_2 : \tau_2$. Mivel $\Gamma, y : \tau', x : \tau_1$ a $\Gamma, x : \tau_1, y : \tau'$ környezet egy permutációja, a 3.4. lemma alapján megkapjuk, hogy $\Gamma, y : \tau', x : \tau_1 \vdash e_2 : \tau_2$, így alkalmazhatjuk a 3.12. szabályt. \square

3.6. 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 -et τ implementációjának, e' -t pedig e kliensének nevezzük. A 3.6. 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 $\Gamma, x : \tau \vdash e_1, e_2 : \text{int}$ 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 3.8. 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 tudjuk, hogy $\Gamma \vdash e_1[x \mapsto e] : \tau_1$ és $\Gamma, y : \tau_1 \vdash e_2[x \mapsto e] : \tau_2$. \square

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

- Nem triviális.
- Típusok unicitása.
- Típusozás inverziója.
- Környezet változóinak sorrendje nem számít.
- Gyengítési tulajdonság.
- Helyettesítés.

3.7. Feladat. *miért nem típusozható az alábbi kifejezés*

3.8. Feladat. *nem szintaxisvezérelt típusrendszer*

3.9. Feladat. *Bizonyítsuk be, hogy ha $\Gamma \vdash e : \tau$, akkor $\Gamma \models e$.*

További információ: a lineáris típusrendszerekben nem teljesül a gyengítési tulajdonság.

3.3. Operációs szemantika

3.4. Szintaxis

3.5. Operációs szemantika

4. Függvények

5. Véges adattípusok

6. Végtelen adattípusok

7. Parcialitás, rekurzív típusok

8. Imperatív programozás

9. Polimorfizmus

10. Altípus

11. Típuselmélet