Nyelvek típusrendszere
(jegyzet)

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

2017. szeptember 13.

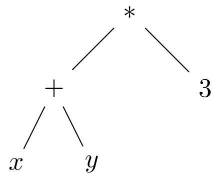
Tartalomjegyzék

1. Bevezető

Típusrendszerek helye a fordítóprogamokban [1, 2]:

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 [7].

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). Ilyenek a függő típusrendszerek, ezeket azonban nem tárgyaljuk. Az érdeklődőknek a homotópia típuselmélet könyv [7] első fejezeteit és az Agda programozási nyelvet [8] javasoljuk.

A típusrendszerekről szól Csörnyei Zoltán könyve [3], a magyar elnevezésekben erre támaszkodunk. A tárgyalás során kisebb részben Pierce-t [6], nagyobb részben Harpert [4] követjük, néhány típusrendszer teljesen megegyezik utóbbi tárgyalásával.

2. Számok és logikai értékek

Ebben a fejezetben egy egyszerű nyelvet tanulmányozunk, mellyel számokból és logikai értékekből álló kifejezéseket tudunk megadni. A fejezet célja a következő alapfogalmak gyakorlása: szintaxis, típusrendszer és operációs szemantika. A nyelv szintaxisa megadja az összes lehetséges kifejezést, a típusrendszer pedig ezt megszorítja az értelmes kifejezésekre (például egy számnak és egy logikai értéknek az összege nem értelmes, de két szám összege már igen). A fejezet végén a nyelv jelentését tanulmányozzuk, megnézzük a denotációs és az operációs szemantikáját. Végül a típusrendszer és az operációs szemantika kapcsolatára fókuszálunk.

2.1. Szintaxis

A szintaxist a következő BNF jelöléssel [5] adjuk meg.

$$
\begin{array}{rlll}
t, t', ... \in \mathsf{Tm} ::= & \mathsf{num}\, n & \text{természetes szám beágyazása} & (2.1)\\
& \mid t + t' & \text{összeadás} & \\
& \mid \mathsf{isZero}\, t & \text{tesztelés} & \\
& \mid \mathsf{true} & \text{igaz logikai érték} & \\
& \mid \mathsf{false} & \text{hamis logikai érték} & \\
& \mid \mathsf{if}\, t\, \mathsf{then}\, t'\, \mathsf{else}\, t'' & \text{elágazás} &
\end{array}
$$

Tm-et (term) *fajtának* nevezzük, ez egy halmaz, mely kifejezéseket tartalmaz. A $t, t', ...$-t *metaváltozóknak* nevezzük, mivel ezek a *metaélméletünk* változói. Metaelméletnek vagy metanyelvnek nevezzük azt a nyelvet, amiben a jegyzet mondatai, definíciói, bizonyításai íródnak. Ezt megkülönböztetjük az *objektumelmélettől* (objektumnyelvtől), attól a nyelvtől, amiről éppen beszélünk (a metanyelvünkben), jelen esetben a természetes számok és logikai kifejezések nyelvétől. Az objektumnyelvünk amúgy nem is tartalmaz változókat (a 3. fejezetben már olyan objektumnyelvet adunk meg, amiben vannak változók).

Az n is metaváltozó, ezzel egy metaelméleti természetes számot (0,1,2,...) jelölünk, n fajtája metaélméleti természetes szám, t fajtája Tm.

A szintaxist *operátorokkal* építjük fel, Tm fajtájú kifejezéseket hatféle operátorral tudunk létrehozni.

Ha n egy természetes szám, akkor $\mathsf{num}\, n$ term, például $\mathsf{num}\, 0 \in \mathsf{Tm}$ és $\mathsf{num}\, 3 \in \mathsf{Exp}$. Ha t és t' termek, akkor $t + t'$ is egy term, például $\mathsf{num}\, 0 + {} + \mathsf{num}\, 3$, $\mathsf{num}\, 3 + \mathsf{num}\, 0$. Ez a két kifejezés nem egyenlő (nem ugyanaz), bár a jelentésük majd meg fog egyezni (lásd 2.3. fejezet). Ha t egy term, akkor $\mathsf{isZero}\, t$ is term, például $\mathsf{isZero}\, (\mathsf{num}\, 2)$ vagy $\mathsf{isZero}\, (\mathsf{num}\, 0 + \mathsf{num}\, 3)$. true és false termek, és ha van három termünk, akkor ebből a háromból az $\mathsf{if} - \mathsf{then} - \mathsf{else} -$ operátor segítségével újabb termet tudunk építeni.

Az operátorokat *aritásukkal* jellemezzük, ez megadja, hogy milyen fajtájú paramétereket várnak, és milyen fajtájú kifejezést adnak eredményül. A fenti hat operátor aritásai:

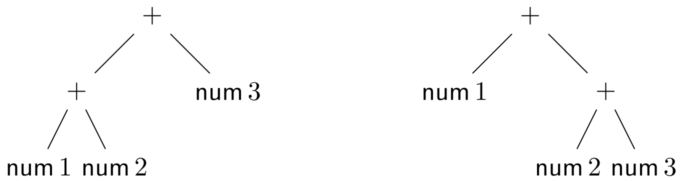
$$
\begin{array}{ll}
\mathsf{num} & (\mathbb{N})\mathsf{Tm}\\
- + - & (\mathsf{Tm}, \mathsf{Tm})\mathsf{Tm}\\
\mathsf{isZero} & (\mathsf{Tm})\mathsf{Tm}\\
\mathsf{true} & ()\mathsf{Tm}\\
\mathsf{false} & ()\mathsf{Tm}\\
\mathsf{if} - \mathsf{then} - \mathsf{else} - & (\mathsf{Tm}, \mathsf{Tm}, \mathsf{Tm})\mathsf{Tm}
\end{array}
$$

$-$-el jelöljük az infix operátorok paramétereinek helyeit. num és isZero prefix operátorok, ezek paramétereit egyszerűen az operátor után írjuk. Az aritásban a zárójelben a paraméterek fajtái vannak felsorolva (vesszővel elválasztva), majd az operátor fajtája következik. true-t és false-ot nulláris operátornak nevezzük, num és isZero unáris, $- + -$ bináris, $\mathsf{if} - \mathsf{then} - \mathsf{else} -$ pedig ternáris operátor.

Az operátorok aritásai a szintaxis BNF definíciójából leolvashatók.

A $+$ operátort ne keverjük össze a metaelméleti összeadással.

A kifejezések valójában fák (absztrakt szintaxisfa), csak a rövidség miatt írjuk őket lineárisan. A zárójelekkel jelezzük, hogy pontosan melyik fáról van szó. Az alábbi bal oldali kifejezés $(\mathsf{num}\,1 + \mathsf{num}\,2) + \mathsf{num}\,3$, a jobb oldali pedig $\mathsf{num}\,1 + (\mathsf{num}\,2 + \mathsf{num}\,3)$.



Ebben a tantárgyban a szintaxissal ezen az absztakciós szinten foglalkozunk: nem foglalkozunk a parsing-al (szintaktikus elemzés [2], tehát, hogy hogyan lesz egy sztringből absztrakt szintaxisfa).

Nem minden term értelmes, például $\mathsf{isZero}\,\mathsf{true}$ és $\mathsf{true} + \mathsf{num}\,3$ termek, de a jelentésük nem világos. Az ilyen értelmetlen termek kiszűrésére való a típusrendszer (2.2. fejezet). A $\mathsf{num}\,(\mathsf{num}\,3 + \mathsf{num}\,2)$ nem kifejezés, mert a num operátor paramétere természetes szám kell, hogy legyen, nem term. A $\mathsf{num}\,3 + 2$ nem term, mert a 2 nem term.

2.1. Feladat. *Ha kiegészítenénk a 2.1 szintaxist az alábbi operátorokkal, mi lenne az aritásuk?*

$$
\begin{aligned}
t, t', ... \in \mathsf{Tm} ::= & \; ... \\
& \mid t * t' \\
& \mid \mathsf{not}\,t \\
& \mid \mathsf{and}\,t\,t'
\end{aligned}
$$

2.2. Feladat. *Döntsd el, hogy termek -e az alábbi kifejezések!*

a) $\mathsf{if}\,\mathsf{num}\,3\,\mathsf{then}\,\mathsf{true}\,\mathsf{else}\,\mathsf{num}\,4$

b) $\mathsf{if}\,\mathsf{true}\,\mathsf{then}\,\mathsf{true}\,\mathsf{else}\,\mathsf{num}\,4$

c) $\mathsf{if}\,\mathsf{true}\,\mathsf{then}\,\mathsf{num}\,3\,\mathsf{else}\,\mathsf{num}\,4$

d) $\mathsf{if}\,3\,\mathsf{then}\,\mathsf{num}\,3\,\mathsf{else}\,4$

e) $\mathsf{if}\,(\mathsf{if}\,\mathsf{true}\,\mathsf{then}\,\mathsf{true}\,\mathsf{else}\,\mathsf{num}\,2)\,\mathsf{then}\,\mathsf{num}\,3\,\mathsf{else}\,\mathsf{true}$

f) $1 + (2 + \mathsf{true})$

g) $\mathsf{num}\,1 + (\mathsf{num}\,2 + \mathsf{true})$

2.1.1. Term szerinti rekurzió és indukció

A fenti Tm egy induktív halmaz, tehát a legkisebb olyan halmaz, mely zárt a $\mathsf{num}, - + -, \mathsf{isZero}$ stb. operátorokra. Az induktív halmazokból *rekurzióval* (strukturális rekurzió) függvényeket tudunk megadni, ill. *indukcióval* (strukturális indukció) bizonyítani tudunk róluk valamit.

Rekurzióval tudjuk megadni például a $height \in \mathsf{Tm} \to \mathbb{N}$ függvényt, mely megállapítja a szintaxisfa magasságát. Ehhez elég megadni, hogy a függvény milyen eredményt ad a különböző operátorokon, felhasználva, hogy tudjuk, hogy milyen eredményt ad az operátorok paraméterein. Az $:=$ jel bal oldalán megadjuk a függvény nevét, és hogy melyik operátorra alkalmazzuk, a jobb oldalán pedig az eredményt, amelyben felhasználhatjuk a függvény hívásának eredményét a paraméteren (ez a rekurzív hívás, ezért nevezzük ezt a megadási módot rekurziónak).

$$\begin{aligned}
&height(\mathsf{num}\,n) &&:= 0\\
&height(t+t') &&:= 1 + max\{height(t), height(t')\}\\
&height(\mathsf{isZero}\,t) &&:= 1 + height(t)\\
&height(\mathsf{true}) &&:= 0\\
&height(\mathsf{false}) &&:= 0\\
&height(\mathsf{if}\,t\,\mathsf{then}\,t'\,\mathsf{else}\,t'') &&:= 1 + max\{height(t), height(t)', height(t'')\}
\end{aligned}$$

Vegyük észre, hogy a $+$ függvény, amit az $:=$ jobb oldalán használunk, a metaelméleti összeadás, míg a bal oldalon az objektumelméleti összeadás operátor szerepel $(t + t')$. max-al jelöljük a metaélméleti maximum függvényt.

A következő függvény megadja, hogy hány true szerepel egy termben:

$$\begin{aligned}
&trues(\mathsf{num}\,n) &&:= 0\\
&trues(t+t') &&:= trues(t) + trues(t')\\
&trues(\mathsf{isZero}\,t) &&:= trues(t)\\
&trues(\mathsf{true}) &&:= 1\\
&trues(\mathsf{false}) &&:= 0\\
&trues(\mathsf{if}\,t\,\mathsf{then}\,t'\,\mathsf{else}\,t'') &&:= trues(t) + trues(t') + trues(t'')
\end{aligned}$$

Vegyük észre, hogy megint használtuk a metaelméleti összeadást.

2.3. Feladat. *Add meg rekurzióval a* $size \in \mathsf{Tm} \to \mathbb{N}$ *függvényt, mely megadja, hogy hány operátor van felhasználva a termben. Például* $size((\mathsf{num}\,1 + \mathsf{true}) + + \mathsf{true}) = 5$*,* $size(\mathsf{isZero}\,\mathsf{true}) = 2$*.*

Ha szeretnénk valamilyen állítást minden termről belátni, azt indukcióval tudjuk megtenni. Ez a természetes számokon alkalmazott teljes indukció általánosítása termekre. Ha a $P(t)$ állítást szeretnénk belátni minden termre, akkor a következőket kell belátnunk:

– minden n-re $P(\mathsf{num}\,n)$,

– ha valamely t, t'-re igaz $P(t)$ és $P(t')$, akkor igaz $P(t + t')$ is,

– ha $P(t)$, akkor $P(\mathsf{isZero}\,t)$,

– igaz $P(\mathsf{true})$,

– igaz $P(\mathsf{false})$,

- ha $P(t)$, $P(t')$ és $P(t'')$, akkor $P(\mathsf{if}\, t\, \mathsf{then}\, t'\, \mathsf{else}\, t'')$.

Ha mind a hat esetet beláttuk, akkor tudjuk, hogy minden t-re $P(t)$.

Például lássuk be, hogy minden termre a benne levő true-k száma kisebb, vagy egyenlő, mint három felemelve a term magassága hatványra. Az intuitív magyarázat az, hogy az if – then – else – operátornak van a legtöbb (három) paramétere, emiatt, ha egy kifejezést csak ezzel építünk fel, és egy teljes ternáris fát építünk, akkor annak maximum három a magasságadikon számú levele lehet, amelyekbe true-kat tudunk elhelyezni. Most lássuk be ezt ténylegesen is, indukcióval, az állítás, amit be szeretnénk látni:

$$P(t) = \mathit{trues}(t) \leq 3^{\mathit{height}(t)}.$$

Megnézzük az összes esetet:

- Minden n-re $\mathit{trues}(\mathsf{num}\, n) \leq 3^{\mathit{height}(\mathsf{num}\, n)}$: a bal oldal a trues definíciója alapján 0, ez pedig minden számnál kisebb vagy egyenlő.

- Tudjuk, hogy $\mathit{trues}(t) \leq 3^{\mathit{height}(t)}$ és $\mathit{trues}(t') \leq 3^{\mathit{height}(t')}$. Ekkor azt szeretnénk belátni, hogy

  $$\mathit{trues}(t + t') \leq (\mathit{height}(t + t'))^3.$$

  Ha $\mathit{height}(t) \leq \mathit{height}(t')$, akkor a következő érvelést alkalmazzuk (a jobb oldalra van írva az adott lépés indoklása).

  $$\begin{aligned}
  & \mathit{trues}(t + t') && \\
  & && \mathit{trues} \text{ definíciója} \\
  = {} & \mathit{trues}(t) + \mathit{trues}(t') && \\
  & && \text{indukciós feltevés (amit tudunk)} \\
  \leq {} & 3^{\mathit{height}(t)} + 3^{\mathit{height}(t')} && \\
  & && \text{feltettük, hogy } \mathit{height}(t) \leq \mathit{height}(t') \\
  \leq {} & 2 * 3^{\mathit{height}(t')} && \\
  & && \text{középiskolai matek} \\
  = {} & 3 * 3^{\mathit{height}(t')} && \\
  & && \text{középiskolai matek} \\
  = {} & 3^{1+\mathit{height}(t')} && \\
  & && \text{feltettük, hogy } \mathit{height}(t) \leq \mathit{height}(t') \\
  = {} & 3^{1+\mathit{max}\{\mathit{height}(t), \mathit{height}(t')\}} && \\
  & && \mathit{height} \text{ definíciója} \\
  = {} & 3^{\mathit{height}(t+t')} &&
  \end{aligned}$$

  A $\mathit{height}(t) > \mathit{height}(t')$ eset hasonlóan belátható.

- Tudjuk, hogy $\mathit{trues}(t) \leq 3^{\mathit{height}(t)}$. Szeretnénk belátni, hogy $\mathit{trues}(\mathsf{isZero}\, t) \leq$

$\leq 3^{height(\mathsf{isZero}\,t)}$. Ezt az alábbi egyszerű érveléssel látjuk:

$$\begin{aligned}
& trues\,(\mathsf{isZero}\,t) \\
= {} & trues\,t \\
\leq {} & 3^{height\,t} \\
\leq {} & 3 * 3^{height\,t} \\
= {} & 3^{1+height\,t} \\
= {} & 3^{height\,(\mathsf{isZero}\,t)}
\end{aligned}$$

- $trues(\mathsf{true}) = 1 \leq 1 = 3^0 = 3^{height(\mathsf{true})}$
- $trues(\mathsf{false}) = 0 \leq 1 = 3^0 = 3^{height(\mathsf{false})}$
- Tudjuk, hogy $trues(t) \leq 3^{height(t)}$, $trues(t') \leq 3^{height(t')}$ és $trues(t'') \leq$ $\leq 3^{height(t'')}$. Feltételezzük, hogy $max\{height(t), height(t'), height(t'')\} =$ $= height(t)$.

$$\begin{aligned}
& trues(\mathsf{if}\,t\,\mathsf{then}\,t'\,\mathsf{else}\,t'') \\
= {} & trues(t) + trues(t') + trues(t'') \\
\leq {} & 3^{height(t)} + 3^{height(t')} + 3^{height(t'')} \\
\leq {} & 3 * 3^{height(t)} \\
= {} & 3^{1+height(t)} \\
= {} & 3^{1+max\{height(t),height(t'),height(t'')\}} \\
= {} & 3^{height(\mathsf{if}\,t\,\mathsf{then}\,t'\,\mathsf{else}\,t'')}
\end{aligned}$$

A $max\{height(t), height(t'), height(t'')\} = height(t')$ és a $max\{height(t), height(t'), height(t'')\} =$ $= height(t'')$ esetek hasonlóak.

2.4. Feladat. *Bizonyítsd be, hogy minden t-re $height(t) < size(t)$!*

2.5. Feladat. *Tekintsük a következő szintaxist (csak természetes számok vannak benne).*

$$t, t', ... \in \mathsf{Tm} ::= \mathsf{num}\,n \mid t + t' \mid t * t'$$

Adjunk meg rekurzióval egy $eval \in \mathsf{Tm} \to \mathbb{N}$ függvényt, mely kiértékeli a kifejezéseket. Például $eval(\mathsf{num}\,3 + (\mathsf{num}\,2 * \mathsf{num}\,4)) = 11$.

2.6. Feladat. *Készítsünk term-optimalizálást: ha van egy $\mathsf{num}\,n + \mathsf{num}\,n'$ részterm egy termben, azt szeretnénk helyettesíteni $\mathsf{num}\,(n + n')$-vel, ahol $+$ a metaelméleti összeadás. Adjunk meg egy $\mathsf{Tm} \to \mathsf{Tm}$ függvényt, mely ezt hajta végre!*

2.1.2. További információ

Az induktív halmazok megadásával általánosságban az univerzális algebra foglalkozik, mi a 10.2. fejezetben foglalkozunk velük.

2.2. Típusrendszer

A típusrendszer megszorítja a kifejezéseket értelmes kifejezésekre, kiszűri az olyan kifejezéseket, mint az $\mathsf{isZero}\,\mathsf{true}$. A módszer az, hogy a termeket különböző *típusokba* soroljuk, és megadjuk, hogy a különböző operátorok milyen típusú paramétereket fogadnak. Az isZero például csak szám típusú paramétert fogadhat, és logikai típusú kifejezést ad eredményül. Amikor a típusellenőrző azt látja, hogy logikai típusú paraméter van az isZero-nak megadva, akkor hibát jelez.

Az előző fejezetben bevezetett nyelvhez két féle típust adunk meg, és ezt az alábbi BNF definícióval megadott Ty (type) fajtával fejezzük ki.

$$\begin{aligned} A, A', B, ... \in \mathsf{Ty} ::= {} & \mathsf{Nat} && \text{természetes számok típusa} \\ \mid {} & \mathsf{Bool} && \text{logikai értékek típusa} \end{aligned} \tag{2.2}$$

Ty egy nagyon egyszerű induktív halmaz, csak két eleme van (két nulláris operátor), Nat és Bool. $A, A', B, ...$ Ty fajtájú metaváltozók. Ty-on úgy tudunk rekurzív függvényt megadni, hogy megadjuk, mi az eredménye Nat-on és hogy mi az eredménye Bool-on. Hasonlóképp, ha valamit bizonyítani szeretnénk minden Ty-ról, elég, ha belátdjuk Nat-ra és Bool-ra.

A *típusozási reláció* egy bináris reláció a termek és a típusok között, jelölése $- : - \subseteq \mathsf{Tm} \times \mathsf{Nat}$. Ha $t \in \mathsf{Tm}$ és $A \in \mathsf{Ty}$, akkor $t : A$-t *ítéletnek* nevezzük, ez attól függően igaz, hogy relációban áll -e t és A. Intuitívan $t : A$ azt mondja, hogy a t termnek A típusa van.

A termek és a típusok induktív halmazok, és az operátoraikkal adtuk meg őket. A típusozási relációt is induktívan adjuk meg, a *levezetési szabályaival* (szabályok, típusozási szabályok, inference rules, typing rules). A 2.1 szintaxis típusozási relációját a következő levezetési szabályokkal adjuk meg.

$$\frac{}{\mathsf{num}\,n : \mathsf{Nat}} \tag{2.3}$$

$$\frac{t : \mathsf{Nat} \qquad t' : \mathsf{Nat}}{t + t' : \mathsf{Nat}} \tag{2.4}$$

$$\frac{t : \mathsf{Nat}}{\mathsf{isZero}\,t : \mathsf{Bool}} \tag{2.5}$$

$$\frac{}{\mathsf{true} : \mathsf{Bool}} \tag{2.6}$$

$$\frac{}{\mathsf{false} : \mathsf{Bool}} \tag{2.7}$$

$$\frac{t : \mathsf{Bool} \qquad t' : A \qquad t'' : A}{\mathsf{if}\,t\,\mathsf{then}\,t'\,\mathsf{else}\,t'' : A} \tag{2.8}$$

Minden operátorhoz, ami a szintaxisban van, tartozik egy levezetési szabály. A vízszintes vonal fölött vannak a szabály feltételei, alatta pedig a konklúzió. Tehát például tetszőleges n (metaelméleti) természetes számra le tudjuk vezetni, hogy a $\mathsf{num}\,n$ term típusa Nat. Ha van két természetes szám típusú termünk, akkor ezekre alkalmazva a $+$ operátort, egy természetes szám fajtájú termet kapunk. Az isZero operátor egy Nat típusú termből Bool típusú termet készít. A true és a false operátorok Bool típusú termeket hoznak létre. Azokat a levezetési szabályokat, melyeknek nincs feltételük, *axiómáknak* nevezzük, ilyenek a 2.6 és

2.7 szabályok. A 2.8 szabály azt fejezi ki, hogy ha van egy Bool típusú termünk, és két termünk, melyeknek ugyanaz az A típusa van, akkor az $\mathsf{if} - \mathsf{then} - \mathsf{else} -$ operátort alkalmazva kapunk egy új A típusú termet.

A fenti szabályok valójában szabály-sémák, vagyis a bennük szereplő metaváltozók minden lehetséges értékére van egy szabályunk. Például a 2.8 szabály egy speciális esete az alábbi.

$$\frac{\mathsf{true} : \mathsf{Bool} \qquad \mathsf{num}\,1 : \mathsf{Nat} \qquad \mathsf{num}\,0 : \mathsf{Nat}}{\mathsf{if}\,\mathsf{true}\,\mathsf{then}\,\mathsf{num}\,1\,\mathsf{else}\,\mathsf{num}\,0 : \mathsf{Nat}}$$

Fontos, hogy ugyanazokat a metaváltozókat ugyanazokra az értékekre kell helyettesíteni, a 2.8 szabálynak nem speciális esete a következő.

$$\frac{\mathsf{true} : \mathsf{Bool} \qquad \mathsf{num}\,1 : \mathsf{Nat} \qquad \mathsf{num}\,0 : \mathsf{Nat}}{\mathsf{if}\,\mathsf{true}\,\mathsf{then}\,\mathsf{num}\,0\,\mathsf{else}\,\mathsf{num}\,1 : \mathsf{Nat}}$$

Tetszőleges t és A akkor állnak relációban, ha a $t : A$ ítélet *levezethető*. A levezetési szabályokat *levezetési fákká* kombináljuk, és így kapunk levezetéseket. A következő fa azt vezeti le, hogy $\mathsf{isZero}\,(\mathsf{num}\,1 + \mathsf{num}\,2) : \mathsf{Bool}$.

$$\dfrac{\dfrac{\dfrac{}{\mathsf{num}\,1 : \mathsf{Nat}}\,2.3 \qquad \dfrac{}{\mathsf{num}\,2 : \mathsf{Nat}}\,2.3}{\mathsf{num}\,1 + \mathsf{num}\,2 : \mathsf{Nat}}\,2.4}{\mathsf{isZero}\,(\mathsf{num}\,1 + \mathsf{num}\,2) : \mathsf{Bool}}\,2.5$$

A levezetési fákat pont fordítva rajzoljuk, mint a szintaxisfákat: itt a gyökér legalul van, a szintaxisfánál legfölül. Az egyes vízszintes vonalak mellé (a fa csomópontjaira) odaírjuk az alkalmazott szabályt.

Ha azt írjuk, hogy $t : A$, ez azt jelenti, hogy $t : A$ levezethető (létezik olyan levezetési fa, melynek $t : A$ van a gyökerénél). Egy olyan t termet, melyhez létezik egy olyan A, hogy $t : A$, *jól típusozott* termnek nevezzük (típusozható, well-typed term).

Vegyük észre, hogy nem tudjuk levezetni az $\mathsf{isZero}\,\mathsf{true} : \mathsf{Bool}$ ítéletet, mert a 2.5 szabály alkalmazása után elakadunk, a 2.3–2.8 levezetési szabályokon végignézve nincs olyan szabály, mellyel a $\mathsf{true} : \mathsf{Nat}$ levezethető lenne.

$$\dfrac{\dfrac{?}{\mathsf{true} : \mathsf{Nat}}}{\mathsf{isZero}\,\mathsf{true} : \mathsf{Bool}}\,2.5$$

A levezetési fák leveinél mindig axiómák vannak.

2.7. Feladat. *Levezethetők -e az alábbi ítéletek? Rajzoljuk fel a levezetési fát, és jelöljük, ha elakadtunk!*

$$\mathsf{isZero}\,(\mathsf{num}\,0 + \mathsf{num}\,1) : \mathsf{Bool}$$
$$\mathsf{if}\,\mathsf{true}\,\mathsf{then}\,\mathsf{false}\,\mathsf{else}\,\mathsf{false} : \mathsf{Bool}$$
$$\mathsf{num}\,0 + \mathsf{if}\,\mathsf{false}\,\mathsf{then}\,\mathsf{num}\,0\,\mathsf{else}\,\mathsf{num}\,2 : \mathsf{Nat}$$
$$\mathsf{num}\,0 + \mathsf{if}\,\mathsf{false}\,\mathsf{then}\,\mathsf{true}\,\mathsf{else}\,\mathsf{true} : \mathsf{Bool}$$
$$\mathsf{if}\,(\mathsf{isZero}\,(\mathsf{num}\,0 + \mathsf{num}\,1))\,\mathsf{then}\,\mathsf{false}\,\mathsf{else}\,\mathsf{num}\,2 : \mathsf{Nat}$$
$$\mathsf{if}\,(\mathsf{isZero}\,(\mathsf{num}\,0 + \mathsf{num}\,0))\,\mathsf{then}\,\mathsf{false}\,\mathsf{else}\,\mathsf{num}\,2 : \mathsf{Bool}$$

Vegyük észre, hogy az aritás és a típusozási szabály különböző fogalmak. Minden operátornak van aritása, és tartozik hozzá típusozási szabály is: előbbi alacsonyabb szintű fogalom, csak a fajtákra vonatkozik.

2.2.1. Levezetés szerinti rekurzió és indukció

Ahogy a termeken, a levezetéseken is tudunk rekurzióval függvényt megadni. Ezt *levezetés szerinti rekurziónak* nevezzük. Ilyenkor minden egyes levezetési szabályra kell megadnunk, hogy milyen eredményt ad a függvény. Például megadjuk a $[\![-]\!]$ függvényt, amely egy levezetésből készít egy $\mathbb{N}$-beli vagy $\{0,1\}$-beli elemet attól függően, hogy a típus Nat vagy Bool volt.

$$[\![t : \mathsf{Nat}]\!] \in \mathbb{N} \qquad\qquad [\![t : \mathsf{Bool}]\!] \in \{0,1\}$$

A függvényt most minden egyes levezetési szabályra kell megadni, tehát megadjuk, hogy a szabály konklúziójára mi legyen az eredménye, felhasználva a függvény eredményeit a szabály feltételeire.

$$\begin{array}{ll}
[\![\mathsf{num}\, n : \mathsf{Nat}]\!] & := n \\
[\![t + t' : \mathsf{Nat}]\!] & := [\![t : \mathsf{Nat}]\!] + [\![t' : \mathsf{Nat}]\!] \\
[\![\mathsf{isZero}\, t : \mathsf{Bool}]\!] & := 1, \text{ ha } [\![t : \mathsf{Nat}]\!] = 0 \\
 & \ 0, \text{ egyébként} \\
[\![\mathsf{true} : \mathsf{Bool}]\!] & := 1 \\
[\![\mathsf{false} : \mathsf{Bool}]\!] & := 0 \\
[\![\mathsf{if}\, t\, \mathsf{then}\, t'\, \mathsf{else}\, t'' : A]\!] & := [\![t' : A]\!], \text{ ha } [\![t : \mathsf{Bool}]\!] = 1 \\
 & \ [\![t'' : A]\!], \text{ egyébként}
\end{array}$$

Ezt a függvényt nevezzük a 2.1. programozási nyelv *denotációs szemantikájának*. Ez egy módszer arra, hogy megadjuk a termek jelentését (szemantikáját). A különböző típusú termeknek természetesen különböző lesz a jelentése, a természetes szám típusú termeket természetes számokkal értelmezzük, a logikai érték típusú termeket pedig 0-val vagy 1-el, a logikai hamist 0-val, a logikai igazat 1-el.

Vegyük észre, hogy a termeken való rekurzióval ezt a függvényt nem tudtuk volna megadni, hiszen akkor meg kellett volna adnunk az összes term, pl. a $\mathsf{num}\, 0+\mathsf{true}$ jelentését is. A levezetés szerinti rekurzióval elég csak a jól típusozott termek jelentését megadnunk, és azt már meg tudjuk tenni.

Ha egy állítást minden típusozható termre szeretnénk belátni, azt *levezetés szerinti indukcióval* tudjuk megtenni. Ez különbözik a term szerkezete szerinti indukciótól, amit a 2.1.1. fejezetben írtunk le.

A $P(t : A)$ állítás függhet a termtől és a típustól is, például

$$P(t : A) = \text{nincs olyan } A' \in \mathsf{Ty}, A' \neq A, \text{ melyre } t : A'.$$

Ezt nevezzük a típusozás unicitásának, lásd 2.2.2. fejezet.

A levezetés szerinti indukció esetén minden levezetési szabályra, amelynek $t : A$ a konklúziója, be kell látnunk, hogy $P(t : A)$, felhasználva, hogy minden $t' : A'$ feltételre igaz, hogy $P(t' : A')$. A mi típusrendszerünkre a következőket kell belátnunk:

- minden n-re $P(\mathsf{num}\, n : \mathsf{Nat})$,
- minden t-re és t'-re, ha $P(t : \mathsf{Nat})$ és $P(t' : \mathsf{Nat})$, akkor $P(t + t' : \mathsf{Nat})$,
- ha $P(t : \mathsf{Nat})$, akkor $P(\mathsf{isZero}\, t : \mathsf{Bool})$,

– $P(\mathsf{true} : \mathsf{Bool})$,

– $P(\mathsf{false} : \mathsf{Bool})$,

– ha $P(t : \mathsf{Bool})$, $P(t' : A)$ és $P(t'' : A)$, akkor $P(\mathsf{if}\, t\, \mathsf{then}\, t'\, \mathsf{else}\, t'' : A)$.

Hogy illusztráljuk ezt a bizonyítási módszer, a fenti állítást (tehát, hogy másik típussal nem típusozható egy term) belátjuk.

– Minden n-re be kell látnunk, hogy nincs olyan $A' \in \mathsf{Ty}, A' \neq \mathsf{Nat}$, melyre $\mathsf{num}\, n : A'$. A' csak Bool lehet, mert a Nat-on kívül csak ez az egy típusunk van. Azt pedig, hogy nem tudjuk levezetni $\mathsf{num}\, n : \mathsf{Bool}$-t, a 2.3–2.8. szabályokon végignézve látjuk: nincs olyan szabály, mely $\mathsf{num}\, n : \mathsf{Bool}$ alakú ítéletet adna eredményül. Vagy a term formája, vagy a típus nem egyezik.

– Tudjuk, hogy nem $t : \mathsf{Bool}$ és nem $t' : \mathsf{Bool}$, és ebből szeretnénk belátni, hogy nem $t + t' : \mathsf{Bool}$. Ha végignézünk a szabályokon, szintén nincs olyan, mely ezt vezetné le (a feltételeket nem is kell használnunk).

– Azt akarjuk belátni, hogy abból, hogy nem $t : \mathsf{Bool}$, nem $\mathsf{isZero}\, t : \mathsf{Nat}$, hasonló módon belátható.

– Nincs olyan szabály, mely azt vezetné le, hogy $\mathsf{true} : \mathsf{Nat}$.

– Nincs olyan szabály, mely azt vezetné le, hogy $\mathsf{false} : \mathsf{Nat}$.

– Ez a legérdekesebb eset. Tudjuk, hogy nem $t : \mathsf{Nat}$, és az $A' \neq A$-ra nem igaz, hogy $t' : A'$ és az sem igaz, hogy $t'' : A'$. Ebből szeretnénk belátni, hogy nem vezethető le $\mathsf{if}\, t\, \mathsf{then}\, t'\, \mathsf{else}\, t'' : A'$. Ezt egyedül a 2.8. szabállyal tudjuk levezetni, az viszont feltételül szabja, hogy $t' : A'$. Ezt viszont az egyik feltétel alapján nem tudjuk levezetni.

Ezzel bebizonyítottuk, hogy minden $t : A$-ra nincs olyan $A' \neq A$, hogy $t : A'$.

2.2.2. A típusrendszer tulajdonságai

Most a típusrendszer néhány tulajdonságát vizsgáljuk meg.

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

2.8. Lemma. *Van olyan t, melyhez nem létezik A, hogy $t : A$.*

Bizonyítás. Például az előbbi $\mathsf{isZero}\, \mathsf{true}$. □

Minden kifejezés maximum egyféleképpen típusozható (*típusok unicitása*).

2.9. Lemma. *Egy t kifejezéshez maximum egy A típus tartozik, melyre $t : A$.*

Bizonyítás. Lásd a 2.2.1. alfejezet végén. □

A típusrendszer ezenkívül *szintaxisvezérelt*: ez azt jelenti, hogy minden kifejezésformát pontosan egy szabály tud levezetni. Ezt fel tudjuk használni ahhoz, hogy megadjunk egy függvényt, mely tetszőleges termnek levezeti a típusát (ha lehetséges). *Típuskikövetkeztetőnek* egy

$$\mathit{infer} \in \mathsf{Tm} \to \mathsf{Ty} \cup \{\mathtt{fail}\}$$

függvényt nevezünk, melyre $t : A$ pontosan akkor, ha $\mathit{infer}(t) = A \in \mathsf{Ty}$.

2.10. Lemma. *Az alábbi infer függvény kikövetkezteti a term típusát.*

$$
\begin{array}{lll}
\mathit{infer}(\mathsf{num}\,n) & := \mathsf{Nat} & \\
\mathit{infer}(t+t') & := \mathsf{Nat} & \textit{, ha } \mathit{infer}(t) = \mathsf{Nat} \textit{ és } \mathit{infer}(t') = \mathsf{Nat} \\
 & \mathtt{fail} & \textit{, egyébként} \\
\mathit{infer}(\mathsf{isZero}\,t) & := \mathsf{Bool} & \textit{, ha } \mathit{infer}(t) = \mathsf{Nat} \\
 & \mathtt{fail} & \textit{, egyébként} \\
\mathit{infer}(\mathsf{true}) & := \mathsf{Bool} & \\
\mathit{infer}(\mathsf{false}) & := \mathsf{Bool} & \\
\mathit{infer}(\mathsf{if}\,t\,\mathsf{then}\,t'\,\mathsf{else}\,t'') & := \mathit{infer}(t') & \textit{, ha } \mathit{infer}(t) = \mathsf{Bool} \textit{ és } \mathit{infer}(t') = \mathit{infer}(t'') \\
 & \mathtt{fail} & \textit{, egyébként}
\end{array}
$$

Bizonyítás. Két lépésben bizonyítunk: először azt, hogy $t : A$-ból következik $\mathit{infer}(t) = A$, utána pedig az ellenkező irányt.

Az odafele irányt megkapjuk $t : A$ levezetés szerinti indukcióval, tehát

$$P(t : A) = (\mathit{infer}(t) = A).$$

A következő eseteket kell megnéznünk:

– Ha a 2.3. szabályt használtuk, akkor a szabály konklúziója miatt $t = \mathsf{num}\,n$, $A = \mathsf{Nat}$, és *infer* definíciója is pont ezt adja.

– Ha a 2.4. szabályt használtuk, akkor $t = t'' + t'$ és $A = \mathsf{Nat}$, és az indukciós feltevéseinkből azt kapjuk, hogy $\mathit{infer}(t'') = \mathsf{Nat}$ és $\mathit{infer}(t') = \mathsf{Nat}$, emiatt pedig *infer* definíciója alapján azt kapjuk, hogy $\mathit{infer}(t'' + t') = \mathsf{Nat}$.

– A 2.5–2.7. esetek hasonlóak.

– Ha a 2.8. szabályt használtuk, akkor a szabály konklúziója miatt $t = \mathsf{if}\,t'''\,\mathsf{then}\,t'\,\mathsf{else}\,t''$, és az indukciós feltevésből azt tudjuk, hogy $\mathit{infer}(t''') = \mathsf{Bool}$, $\mathit{infer}(t') = A$ és $\mathit{infer}(t'') = A$. Ebből és *infer* definíciójából azt megkapjuk, hogy $\mathit{infer}(\mathsf{if}\,t'''\,\mathsf{then}\,t'\,\mathsf{else}\,t'') = A$.

A visszafele irányban t term szerinti indukciót használunk,

$$P(t) = \mathit{infer}(t) = A \in \mathsf{Ty}\text{-ból következik } t : A.$$

Megnézzük az eseteket:

– Ha $t = \mathsf{num}\,n$, akkor $\mathit{infer}(\mathsf{num}\,n) = \mathsf{Nat}$, és a 2.3. szabály alapján $\mathsf{num}\,n : \mathsf{Nat}$.

– Ha $t = t'' + t'$ és $\mathit{infer}(t'' + t') = A \in \mathsf{Ty}$, akkor *infer* definíciója alapján A csak Nat lehet, és szintén *infer* definíciója alapján $\mathit{infer}(t'') = \mathsf{Nat}$ és $\mathit{infer}(t') = \mathsf{Nat}$, az indukciós feltevésből emiatt megkapjuk, hogy $t'' : \mathsf{Nat}$ és $t' : \mathsf{Nat}$, és ezeket a 2.4. szabálynak megadva megkapjuk, hogy $t'' + t' : \mathsf{Nat}$.

– Ha $t = \mathsf{isZero}\,t'$ és $\mathit{infer}(t'') = A \in \mathsf{Ty}$, akkor *infer* definíciója alapján A csak Bool lehet, és ekkor *infer* definíciója alapján tudjuk, hogy $\mathit{infer}(t') = \mathsf{Nat}$, ebből és az indukciós feltevésből megkapjuk, hogy $t' : \mathsf{Nat}$, és ezt a 2.5. szabálynak megadva kapjuk, hogy $\mathsf{isZero}\,t' : \mathsf{Bool}$.

– A többi eset hasonló.

$\square$

2.11. Feladat. *Írjuk le részletesen a 2.10. lemma bizonyításának a maradék eseteit.*

Típusellenőrzőnek egy olyan $\mathit{check} \in \mathsf{Tm} \times \mathsf{Ty} \to \{\mathtt{success}, \mathtt{fail}\}$ függvényt nevezünk, melyre $t : A$ pontosan akkor, ha $\mathit{check}(t, A) = \mathtt{success}$.

2.12. Lemma. *Az alábbi check függvény típusellenőrző.*

$$\begin{aligned}\mathit{check}(t, A) := {} & \mathtt{success}, \text{ ha } \mathit{infer}(t) = A \\ & \mathtt{fail} \quad\ , \text{ egyébként}\end{aligned}$$

2.13. Feladat. *Bizonyítsuk be a 2.12. lemmát.*

2.3. Operációs szemantika (TODO)

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.

$$\frac{}{e \longmapsto^* e} \tag{2.9}$$

$$\frac{e \longmapsto e' \qquad e' \longmapsto^* e''}{e \longmapsto^* e''} \tag{2.10}$$

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.

$$\frac{}{n\,\mathsf{val}} \tag{2.11}$$

$$\frac{}{\text{"}s\text{"}\ \mathsf{val}} \tag{2.12}$$

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

$$\frac{n_1 + n_2 = n}{n_1 + n_2 \longmapsto n} \tag{2.13}$$

$$\frac{n_1 - n_2 = n}{n_1 - n_2 \longmapsto n} \tag{2.14}$$

$$\frac{s_1 \text{ és } s_2 \text{ konkatenáltja } s}{\text{"}s_1\text{"} \bullet \text{"}s_2\text{"} \longmapsto \text{"}s\text{"}} \tag{2.15}$$

$$\frac{s \text{ hossza } n}{|"s"| \longmapsto n} \tag{2.16}$$

$$\frac{e_1 \longmapsto e_1'}{e_1 \circ e_2 \longmapsto e_1' \circ e_2} \circ \in \{+, -, \bullet\} \tag{2.17}$$

$$\frac{e_1 \,\mathsf{val} \qquad e_2 \longmapsto e_2'}{e_1 \circ e_2 \longmapsto e_1 \circ e_2'} \circ \in \{+, -, \bullet\} \tag{2.18}$$

$$\frac{e \longmapsto e'}{|e| \longmapsto |e'|} \tag{2.19}$$

$$\left[\frac{e_1 \longmapsto e_1'}{\mathsf{let}\, e_1 \,\mathsf{in}\, x.e_2 \longmapsto \mathsf{let}\, e_1' \,\mathsf{in}\, x.e_2}\right] \tag{2.20}$$

$$\frac{[e_1 \,\mathsf{val}]}{\mathsf{let}\, e_1 \,\mathsf{in}\, x.e_2 \longmapsto e_2[x \mapsto e_1]} \tag{2.21}$$

A szemantikának két változata van: *érték szerinti* (by value) és *név szerinti* (by name) paraméterátadás. Utóbbinál elhagyjuk a zárójelezett 5.28. szabályt és a zárójelezett feltételt az 5.29. szabályból.

Az 5.21–5.24. és az 5.29. szabályok utasítás szabályok, ezek adják meg, hogy ha egy operátornak már ki vannak értékelve a paraméterei, hogyan adjuk meg az eredményét. Az 5.25–5.28. szabályok sorrendi szabályok, ezek adják meg, hogy milyen sorrendben történjék a kiértékelés. Például a $|"\mathsf{aa}"| + (3 - 2)$ 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:

$$\begin{array}{ll} & |"\mathsf{aa}"| + (3 - 2) \\ 5.25 & \longmapsto 2 + (3 - 2) \\ 5.22, 5.26 & \longmapsto 2 + 1 \\ 5.21 & \longmapsto 3 \end{array}$$

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).

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

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

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

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

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

2.18. 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 az 5.21–5.29. szabályok megtartják P-t.

Például bebizonyítjuk az alábbi lemmát.

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

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

Determináltság.

2.20. 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 $-$ 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.

2.21. Feladat. *Mutassuk meg, hogy bármely* s_1*,* s_2*-re létezik olyan* e*, hogy* $|"s_1" \bullet "s_2"| \longmapsto^* e$ *és* $|"s_1"| + |"s_2"| \longmapsto^* e$*.*

2.4. Típusrendszer és szemantika kapcsolata (TODO)

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.

2.22. Tétel. *Ha* $e : \tau$ *és* $e \longmapsto e'$*, akkor* $e' : \tau$*.*

Bizonyítás. $e \longmapsto e'$ szerinti indukció. Az 5.21. szabály esetén tudjuk, hogy $n_1 + + n_2 \longmapsto n$, és $n_1 + n_2 : \tau$, és az inverziós lemmából (5.4) tudjuk, hogy $\tau = \mathsf{int}$, és azt is tudjuk, hogy $n : \mathsf{int}$. Hasonló a bizonyítás az 5.22–5.24. esetekben. Az 5.25. 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 (5.12) megadja, hogy $e_1' + e_2 : \mathsf{int}$. Az 5.26. esetén ugyanilyen az indoklás, csak a második paraméter változik. Az 5.27. esetén tudjuk, hogy $|e| \longmapsto |e'|$ és az indukciós hipotézisből és az inverzióból kapjuk, hogy $e' : \mathsf{str}$, ebből az 5.15. szabály alapján kapjuk, hogy $|e'| : \mathsf{int}$. Az 5.28. 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 (5.16) alapján kapjuk, hogy $\mathsf{let}\, e_1' \,\mathsf{in}\, x.e_2 : \tau_2$. Az 5.29. 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 (5.7) látjuk be. □

Haladás (progress). Ez a tétel 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.

2.23. Tétel. *Ha $e : \tau$, akkor vagy $e\,\mathsf{val}$, vagy létezik olyan e', hogy $e \longmapsto e'$.*

A bizonyításhoz szükségünk van a következő lemmára a kanonikus alakokról.

2.24. 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. Az 5.9. levezetés nem fordulhat elő, mert a környezet üres. Az 5.10–5.11. esetén már értékeink vannak, és nincs olyan átírási szabály, mely alkalmazható lenne. Az 5.12. 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 $e_2 \longmapsto e_2'$. 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 $e_2 : \mathsf{int}$. Az 5.23. lemmából és ebből kapjuk, hogy $e_1 = n_1$ és $e_2 = n_2$, így ha n_1 és n_2 összege n, akkor az 5.21. szabály szerinti $e_1 + e_2 \longmapsto n$ átírást hajtjuk végre. Az 5.13–5.14. esetben hasonló módon járunk el. Az 5.15. esetben az indukciós feltevésből tudjuk, hogy $e : \mathsf{str}$ és vagy $e\,\mathsf{val}$ vagy $e \longmapsto e'$. Előbbi esetben az 5.23. lemmából kapjuk, hogy $e = {}$"s" valamely s-re, és ha s hossza n, akkor végrehajtjuk a $|$"s"$| \longmapsto n$ átírást (az 5.24. szabály), utóbbi esetben az 5.27. szabály alapján lépünk $|e| \longmapsto |e'|$-t. Az 5.16. 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 (5.29), érték szerinti paraméterátadás esetén az indukciós feltevéstől függően tesszük meg az 5.29. vagy az 5.28. lépést. □

2.5. Futási idejű hibák (TODO)

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

$$\frac{\Gamma \vdash e_1 : \mathsf{int} \qquad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1/e_2 : \mathsf{int}} \tag{2.22}$$

$$\frac{n_1 \text{ osztva } n_2\text{-vel } n}{n_1/n_2 \longmapsto n} \tag{2.23}$$

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 $e\,\mathsf{err}$ ítélettel, ami azt fejezi ki, hogy e kiértékelése hibával végződik. A következő levezetési szabályokkal egészítjük ki az 5.21–5.29. szabályokkal megadott átíró rendszert.

$$\frac{e_1\,\mathsf{val}}{e_1/0\,\mathsf{err}} \tag{2.24}$$

$$\frac{e_1\,\mathsf{err}}{e_1/e_2\,\mathsf{err}} \tag{2.25}$$

$$\frac{e_1\,\mathsf{val} \qquad e_2\,\mathsf{err}}{e_1/e_2\,\mathsf{err}} \tag{2.26}$$

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

$$\frac{e_1\,\mathsf{err}}{e_1 + e_2\,\mathsf{err}} \tag{2.27}$$

$$\frac{e_1\,\mathsf{val} \qquad e_2\,\mathsf{err}}{e_1 + e_2\,\mathsf{err}} \tag{2.28}$$

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

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

$$\mathsf{Exp} ::= \ldots \mid \mathsf{error} \tag{2.29}$$

A típusrendszert a következő szabállyal egészítjük ki.

$$\frac{\Gamma\,\mathsf{wf}}{\Gamma \vdash \mathsf{error} : \tau} \tag{2.30}$$

Az operációs szemantika a következő szabállyal egészül ki.

$$\overline{\mathsf{error}\,\mathsf{err}} \tag{2.31}$$

2.26. Feladat. *Egészítsük ki az 5.22. tételt hibákkal. Tehát bizonyítsuk be, hogy ha $e : \tau$, akkor vagy $e\,\mathsf{err}$, vagy $e\,\mathsf{val}$, vagy létezik olyan e', hogy $e \longmapsto e'$.*

2.27. Feladat. *Egészítsük ki a nyelvet (szintaxist, típusrendszert, operációs szemantikát) egy operátorral, mely egy szövegnek kiveszi az első betűjét (és egy egy karakter hosszú szöveget készít belőle). Ha üres szövegre alkalmazzuk, hibát adjon.*

3. Lambda kalkulus

Típus nélküli.

4. Induktív definíciók

Harper könyv: I. rész.

4.1. Absztrakt szintaxisfák

Absztrakt szintaxisfának (AST, abstract syntax tree) nevezzük az olyan fákat, 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.

$$n, n', ... \in \mathsf{Nat} ::= i \,|\, \mathsf{zero} \,|\, \mathsf{suc}\, n \tag{4.1}$$
$$e, e', ... \in \mathsf{Exp} ::= x \,|\, \mathsf{num}\, n \,|\, e + e' \tag{4.2}$$

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* $()\mathsf{Nat}$) vagy az unáris suc operátorral (aritása $(\mathsf{Nat})\mathsf{Nat}$). n egy tetszőleges Nat fajtájú AST-t jelöl, míg i maga egy Nat fajtájú AST, mely egy darab változóból áll.

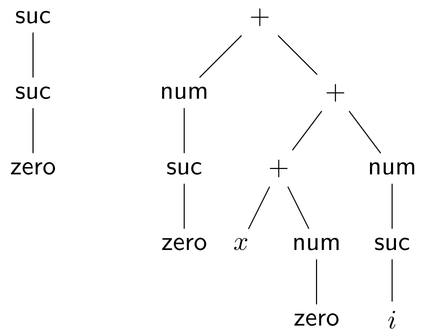
Az Exp fajtájú AST-ket e-vel és ennek vesszőzött változataival jelöljük, az Exp fajtájú változókat x-el jelöljük. Exp fajtájú AST-t egy unáris operátorral (num, aritása $(\mathsf{Nat})\mathsf{Exp}$) és egy bináris operátorral ($+$, aritása $(\mathsf{Exp}, \mathsf{Exp})\mathsf{Exp}$) tudunk létrehozni.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

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

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).

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

Néhány példa:

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

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).

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

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

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

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

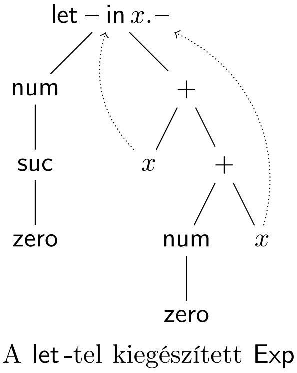
További információ:

– 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 $b \in B_{Y_A,\dots Y_B}$, 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.

4.2. 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 $(\mathsf{Exp}, \mathsf{Exp}.\mathsf{Exp})\mathsf{Exp}$-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 $\mathsf{let\,num\,zero\,in}\, x.x+x$ ABT zárt. A kötések hatásköre a lehető legtovább tart, emiatt

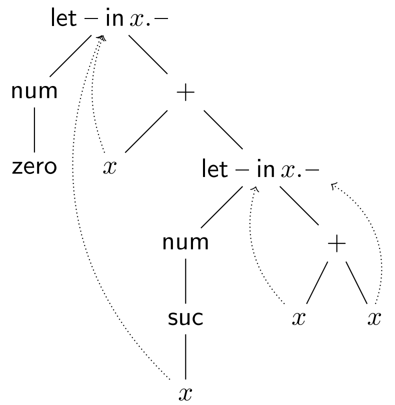
$$\mathsf{let}\, e \,\mathsf{in}\, x.(e_0 + e_1) = \mathsf{let}\, e \,\mathsf{in}\, x.e_0 + e_1 \neq (\mathsf{let}\, e \,\mathsf{in}\, x.e_0) + e_1.$$

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

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

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.

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

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 köt[1].

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

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 $\mathcal{Q}(\mathsf{let}\, e\, \mathsf{in}\, x.e')$.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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).

4.3. Levezetési fák

Ítéleteket ABT-ről 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.

$$\frac{}{\mathsf{zero} + n'\ \mathsf{is}\ n'} \tag{4.3}$$

$$\frac{n + n'\ \mathsf{is}\ n''}{\mathsf{suc}\, n + n'\ \mathsf{is}\ \mathsf{suc}\, n''} \tag{4.4}$$

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 4.3 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.

$$\dfrac{\dfrac{\dfrac{}{\mathsf{zero} + \mathsf{suc\,zero\ is\ suc\,zero}}\,(4.3)}{(\mathsf{suc\,zero}) + \mathsf{suc\,zero\ is\ suc}\,(\mathsf{suc\,zero})}\,(4.4)}{\mathsf{suc}\,(\mathsf{suc\,zero}) + \mathsf{suc\,zero\ is\ suc}\,\big(\mathsf{suc}\,(\mathsf{suc\,zero})\big)}\,(4.4)$$

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.

$$\dfrac{}{\mathsf{zero\ nat}} \qquad (4.5)$$

$$\dfrac{n\ \mathsf{nat}}{\mathsf{suc}\,n\ \mathsf{nat}} \qquad (4.6)$$

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.

$$\dfrac{}{x\ \mathsf{isBalanced\ zero}} \qquad (4.7)$$

$$\dfrac{}{\mathsf{num}\,n\ \mathsf{isBalanced\ zero}} \qquad (4.8)$$

$$\dfrac{e\ \mathsf{isBalanced}\ n \qquad e'\ \mathsf{isBalanced}\ n}{e + e'\ \mathsf{isBalanced\ suc}\,n} \qquad (4.9)$$

Egy példa levezetés.

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

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 $J_1, ..., J_n$ 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 4.3 szabály alapján bármely $n \in \mathsf{Nat}$-ra le tudjuk vezetni, hogy $\mathsf{zero} + n$ is n. Megmutatjuk a másik irányt.

4.9. 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 4.5 szabályt használtuk, akkor $P(\mathsf{zero}\ \mathsf{nat}) = \mathsf{zero} + \mathsf{zero}\ \mathsf{is}\ \mathsf{zero}$-t kell bizonyítanunk, ezt megtesszük a 4.3 szabállyal. Ha a 4.6 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 $P(suc\, n\ \mathsf{nat}) = \mathsf{suc}\, n + \mathsf{zero}\ \mathsf{is}\ \mathsf{suc}\, n$. A 4.4 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.

4.10. 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 4.7 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 4.7 szabályt használjuk újra.
- Ha a 4.8 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 4.8 szabályt újra alkalmazva megkapjuk a kívánt eredményt.
- Ha a 4.9 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 4.9 szabály alapján látunk.

□

4.11. Feladat. *Adjuk meg a* $\mathsf{max}\, n\, n'\ \mathsf{is}\ n''$ *ítélet levezetési szabályait. Az ítélet azt fejezi ki, hogy* n *és* n' Nat*-beli AST-k maximuma* n''*.*

4.12. Feladat. *Ennek segítségével adjuk meg a* $\mathsf{e}\ \mathsf{hasHeight}\ \mathsf{n}$ *ítélet levezetési szabályait, ahol* e *egy* Exp *fajtájú,* n *pedig egy* Nat *fajtájú AST.*

4.13. 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.*

4.14. 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.

5. Számok és szövegek régi

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.

5.1. Szintaxis

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

$$
\begin{aligned}
\tau, \tau', \ldots \in \mathsf{Ty} &::= \mathsf{int} && \text{egész számok típusa} && (5.1)\\
&\quad\mid \mathsf{str} && \text{szövegek típusa}\\
e, e', \ldots \in \mathsf{Exp} &::= x && \text{változó} && (5.2)\\
&\quad\mid n && \text{egész szám beágyazása}\\
&\quad\mid \text{"}s\text{"} && \text{szöveg beágyazása}\\
&\quad\mid e + e' && \text{összeadás}\\
&\quad\mid e - e' && \text{kivonás}\\
&\quad\mid e \bullet e' && \text{összefűzés}\\
&\quad\mid |e| && \text{hossz}\\
&\quad\mid \mathsf{let}\, e \,\mathsf{in}\, x.e' && \text{definíció}
\end{aligned}
$$

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. $\mathsf{let}\,\text{"}\mathtt{ello}\text{"}\,\mathsf{in}\, x.\text{"}\mathtt{h}\text{"} \bullet x \bullet \text{"}\mathtt{b}\text{"} \bullet x$.

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

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

5.2. 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 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.

$$\Gamma, \Gamma', ... \in \mathsf{Con} ::= \cdot \mid \Gamma, x : \tau \tag{5.3}$$

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.

$$\begin{aligned} \mathsf{dom}(\cdot) &:= \{\} \\ \mathsf{dom}(\Gamma, x : \tau) &:= \{x\} \cup \mathsf{dom}(\Gamma) \end{aligned} \tag{5.4}$$

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

$$\frac{}{\cdot\,\mathsf{wf}} \tag{5.5}$$

$$\frac{\Gamma\,\mathsf{wf} \qquad x \notin \mathsf{dom}(\Gamma)}{\Gamma, x : \tau\,\mathsf{wf}} \tag{5.6}$$

Ezután csak jól formált környezetekkel fogunk dolgozni.

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

$$\frac{\Gamma\,\mathsf{wf} \qquad x \notin \mathsf{dom}(\Gamma)}{(x : \tau) \in \Gamma, x : \tau} \tag{5.7}$$

$$\frac{(x : \tau) \in \Gamma \qquad y \notin \mathsf{dom}(\Gamma)}{(x : \tau) \in \Gamma, y : \tau'} \tag{5.8}$$

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.

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x : \tau} \tag{5.9}$$

$$\frac{\Gamma\,\mathsf{wf}}{\Gamma \vdash n : \mathsf{int}} \tag{5.10}$$

$$\frac{\Gamma\,\mathsf{wf}}{\Gamma \vdash \text{"}s\text{"} : \mathsf{str}} \tag{5.11}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{int} \qquad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1 + e_2 : \mathsf{int}} \tag{5.12}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{int} \qquad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1 - e_2 : \mathsf{int}} \tag{5.13}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{str} \qquad \Gamma \vdash e_2 : \mathsf{str}}{\Gamma \vdash e_1 \bullet e_2 : \mathsf{str}} \tag{5.14}$$

$$\frac{\Gamma \vdash e : \mathsf{str}}{\Gamma \vdash |e| : \mathsf{int}} \tag{5.15}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau_2 \qquad x \notin \mathsf{dom}(\Gamma)}{\Gamma \vdash \mathsf{let}\, e_1\, \mathsf{in}\, x.e_2 : \tau_2} \tag{5.16}$$

Példa levezetés:

$$\dfrac{\dfrac{\dfrac{}{\cdot\ \mathsf{wf}}\ 5.5}{\cdot \vdash \text{"a"} : \mathsf{str}}\ 5.11 \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot\ \mathsf{wf}}\ 5.5 \qquad x \notin \{\}}{(x : \mathsf{str}) \in \cdot, x : \mathsf{str}}\ 5.7}{\cdot, x : \mathsf{str} \vdash x : \mathsf{str}}\ 5.9}{\cdot, x : \mathsf{str} \vdash |x| : \mathsf{int}}\ 5.15 \qquad \dfrac{\dfrac{\dfrac{}{\cdot\ \mathsf{wf}}\ 5.5 \qquad x \notin \{\}}{\cdot, x : \mathsf{str}\, \mathsf{wf}}\ 5.6}{\cdot, x : \mathsf{str} \vdash 2 : \mathsf{int}}\ 5.10}{\cdot, x : \mathsf{str} \vdash |x| + 2 : \mathsf{int}}\ 5.12}{\cdot \vdash \mathsf{let}\, \text{"a"}\, \mathsf{in}\, x.|x| + 2 : \mathsf{int}}\ 5.16$$

A $|3|$ kifejezés nem típusozható, mert csak az 5.15. 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.

5.2. 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).

5.3. 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.

5.4. 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. $\square$

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

5.5. 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. Az 5.10–5.15. szabályoknál csak az induktív hipotéziseket, majd a szabályt használjuk. Az 5.16. szabály esetén azt szeretnénk belátni, hogy $\Gamma' \vdash \mathsf{let}\, e_1 \,\mathsf{in}\, x.e_2 : \tau_2$, 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 az 5.16. szabály alapján megkapjuk, hogy $\Gamma' \vdash \mathsf{let}\, e_1 \,\mathsf{in}\, x.e_2 : \tau_2$. Az 5.9. 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 az 5.6 szabályt, ahányadik komponens $(x : \tau)$ hátulról Γ'-ben. $\square$

A típusrendszer egy további fontos tulajdonsága a gyengítési tulajdonság.

5.6. 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 $P(\Gamma \vdash e : \tau) = \Gamma, y : \tau' \vdash e : \tau$, ahol $y \notin \mathsf{dom}(\Gamma)$. A változó esetében eggyel többször használjuk az 5.8. szabályt, az 5.10–5.11. szabályok esetén ugyanazeket a szabályokat alkalmazzuk más környezetekre, az 5.12–5.15. szabályok esetén az indukciós feltevést használjuk, majd magát a szabályt. Az 5.16. 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 $\Gamma, y : \tau' \vdash \mathsf{let}\, e_1 \,\mathsf{in}\, x.e_2 : \tau_2$. Mivel $\Gamma, y : \tau', x : \tau_1$ a $\Gamma, x : \tau_1, y : \tau'$ környezet egy permutációja, az 5.5. lemma alapján megkapjuk, hogy $\Gamma, y : \tau', x : \tau_1 \vdash e_2 : \tau_2$, így alkalmazhatjuk az 5.16. szabályt. $\square$

Helyettesítési lemma.

5.7. 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. Az 5.7. 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. Az 5.10–5.11. szabályok esetén nem csinál semmit a helyettesítés. Az 5.12–5.15. szabályok esetén az indukciós feltevésekből következik a helyettesítés helyessége. Pl. az 5.12. 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 az 5.12. szabály alkalmazásával megkapjuk. Az 5.9 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. Az 5.16. 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 az 5.5. lemma alapján $\Gamma, y : \tau_1, x : \tau \vdash$ $\vdash e_2 : \tau_2$, majd az indukciós feltevés alapján kapjuk, hogy $\Gamma, y : \tau_1 \vdash e_2[x \mapsto e] :$ τ_2. Mivel $(\mathsf{let}\, e_1 \,\mathsf{in}\, y.e_2)[x \mapsto e] = \mathsf{let}\, e_1[x \mapsto e] \,\mathsf{in}\, y.e_2[x \mapsto e]$, az 5.16. 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).

5.8. 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.

- 5.2. lemma: nem triviális.
- 5.3. lemma: típusok unicitása.
- 5.4. lemma: típusozás inverziója.
- 5.5. lemma: környezet változóinak sorrendje nem számít.
- 5.6. lemma: gyengítési tulajdonság.
- 5.7. lemma: helyettesítés.
- 5.8. lemma: dekompozíció.

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

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

5.10. Feladat. *Írjuk fel az 5.4. lemma összes esetét, és bizonyítsuk be őket.*

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

5.12. 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.

5.3. Operációs szemantika

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.

$$\frac{}{e \longmapsto^* e} \tag{5.17}$$

$$\frac{e \longmapsto e' \qquad e' \longmapsto^* e''}{e \longmapsto^* e''} \tag{5.18}$$

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

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

$$\frac{}{n\,\mathsf{val}} \tag{5.19}$$

$$\frac{}{\text{"}s\text{"}\;\mathsf{val}} \tag{5.20}$$

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

$$\frac{n_1 + n_2 = n}{n_1 + n_2 \longmapsto n} \tag{5.21}$$

$$\frac{n_1 - n_2 = n}{n_1 - n_2 \longmapsto n} \tag{5.22}$$

$$\frac{s_1 \text{ és } s_2 \text{ konkatenáltja } s}{\text{"}s_1\text{"} \bullet \text{"}s_2\text{"} \longmapsto \text{"}s\text{"}} \tag{5.23}$$

$$\frac{s \text{ hossza } n}{|\text{"}s\text{"}| \longmapsto n} \tag{5.24}$$

$$\frac{e_1 \longmapsto e_1'}{e_1 \circ e_2 \longmapsto e_1' \circ e_2} \circ \in \{+, -, \bullet\} \tag{5.25}$$

$$\frac{e_1 \,\mathsf{val} \qquad e_2 \longmapsto e_2'}{e_1 \circ e_2 \longmapsto e_1 \circ e_2'} \circ \in \{+, -, \bullet\} \tag{5.26}$$

$$\frac{e \longmapsto e'}{|e| \longmapsto |e'|} \tag{5.27}$$

$$\left[\frac{e_1 \longmapsto e_1'}{\mathsf{let}\, e_1 \,\mathsf{in}\, x.e_2 \longmapsto \mathsf{let}\, e_1' \,\mathsf{in}\, x.e_2}\right] \tag{5.28}$$

$$\frac{[e_1 \,\mathsf{val}]}{\mathsf{let}\, e_1 \,\mathsf{in}\, x.e_2 \longmapsto e_2[x \mapsto e_1]} \tag{5.29}$$

A szemantikának két változata van: *érték szerinti* (by value) és *név szerinti* (by name) paraméterátadás. Utóbbinál elhagyjuk a zárójelezett 5.28. szabályt és a zárójelezett feltételt az 5.29. szabályból.

Az 5.21–5.24. és az 5.29. szabályok utasítás szabályok, ezek adják meg, hogy ha egy operátornak már ki vannak értékelve a paraméterei, hogyan adjuk meg az eredményét. Az 5.25–5.28. szabályok sorrendi szabályok, ezek adják meg, hogy milyen sorrendben történjék a kiértékelés. Például a $|\text{"}\mathsf{aa}\text{"}| + (3 - 2)$ 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:

$$\begin{array}{ll} & |\text{"}\mathsf{aa}\text{"}| + (3 - 2) \\ 5.25 & \longmapsto 2 + (3 - 2) \\ 5.22, 5.26 & \longmapsto 2 + 1 \\ 5.21 & \longmapsto 3 \end{array}$$

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).

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

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

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

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

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

5.17. 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 az 5.21–5.29. szabályok megtartják P-t.

Például bebizonyítjuk az alábbi lemmát.

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

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

Determináltság.

5.19. 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 $-$ 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.

5.20. Feladat. *Mutassuk meg, hogy bármely* s_1*,* s_2*-re létezik olyan* e*, hogy* $|"s_1" \bullet "s_2"| \longmapsto^* e$ *és* $|"s_1"| + |"s_2"| \longmapsto^* e$*.*

5.4. 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.

5.21. Tétel. *Ha* $e : \tau$ *és* $e \longmapsto e'$*, akkor* $e' : \tau$*.*

Bizonyítás. $e \longmapsto e'$ szerinti indukció. Az 5.21. szabály esetén tudjuk, hogy $n_1 + n_2 \longmapsto n$, és $n_1 + n_2 : \tau$, és az inverziós lemmából (5.4) tudjuk, hogy $\tau = \mathsf{int}$, és azt is tudjuk, hogy $n : \mathsf{int}$. Hasonló a bizonyítás az 5.22–5.24. esetekben. Az 5.25. 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 (5.12) megadja, hogy $e_1' + e_2 : \mathsf{int}$. Az 5.26. esetén ugyanilyen az indoklás, csak a második paraméter változik. Az 5.27. esetén tudjuk, hogy $|e| \longmapsto |e'|$ és az indukciós hipotézisből és az inverzióból kapjuk, hogy $e' : \mathsf{str}$, ebből az 5.15. szabály alapján kapjuk, hogy $|e'| : \mathsf{int}$. Az 5.28. 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 (5.16) alapján kapjuk, hogy $\mathsf{let}\, e_1' \,\mathsf{in}\, x.e_2 : \tau_2$. Az 5.29. 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 (5.7) látjuk be. □

Haladás (progress). Ez a tétel 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.

5.22. Tétel. *Ha $e : \tau$, akkor vagy $e\,\mathsf{val}$, vagy létezik olyan e', hogy $e \longmapsto e'$.*

A bizonyításhoz szükségünk van a következő lemmára a kanonikus alakokról.

5.23. 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. Az 5.9. levezetés nem fordulhat elő, mert a környezet üres. Az 5.10–5.11. esetén már értékeink vannak, és nincs olyan átírási szabály, mely alkalmazható lenne. Az 5.12. 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 $e_2 \longmapsto e_2'$. 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 $e_2 : \mathsf{int}$. Az 5.23. lemmából és ebből kapjuk, hogy $e_1 = n_1$ és $e_2 = n_2$, így ha n_1 és n_2 összege n, akkor az 5.21. szabály szerinti $e_1 + e_2 \longmapsto n$ átírást hajtjuk végre. Az 5.13–5.14. esetben hasonló módon járunk el. Az 5.15. esetben az indukciós feltevésből tudjuk, hogy $e : \mathsf{str}$ és vagy $e\,\mathsf{val}$ vagy $e \longmapsto e'$. Előbbi esetben az 5.23. lemmából kapjuk, hogy $e =$ "s" valamely s-re, és ha s hossza n, akkor végrehajtjuk a $|$"s"$| \longmapsto n$ átírást (az 5.24. szabály), utóbbi esetben az 5.27. szabály alapján lépünk $|e| \longmapsto |e'|$-t. Az 5.16. 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 (5.29), érték szerinti paraméterátadás esetén az indukciós feltevéstől függően tesszük meg az 5.29. vagy az 5.28. lépést. □

5.5. 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.

$$\frac{\Gamma \vdash e_1 : \mathsf{int} \qquad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1/e_2 : \mathsf{int}} \tag{5.30}$$

$$\frac{n_1 \text{ osztva } n_2\text{-vel } n}{n_1/n_2 \longmapsto n} \tag{5.31}$$

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 $e\,\mathsf{err}$ ítélettel, ami azt fejezi ki, hogy e kiértékelése hibával végződik. A következő levezetési szabályokkal egészítjük ki az 5.21–5.29. szabályokkal megadott átíró rendszert.

$$\frac{e_1\,\mathsf{val}}{e_1/0\,\mathsf{err}} \tag{5.32}$$

$$\frac{e_1\,\mathsf{err}}{e_1/e_2\,\mathsf{err}} \tag{5.33}$$

$$\frac{e_1\,\mathsf{val} \qquad e_2\,\mathsf{err}}{e_1/e_2\,\mathsf{err}} \tag{5.34}$$

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

$$\frac{e_1\,\mathsf{err}}{e_1 + e_2\,\mathsf{err}} \tag{5.35}$$

$$\frac{e_1\,\mathsf{val} \qquad e_2\,\mathsf{err}}{e_1 + e_2\,\mathsf{err}} \tag{5.36}$$

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

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

$$\mathsf{Exp} ::= \ldots \mid \mathsf{error} \tag{5.37}$$

A típusrendszert a következő szabállyal egészítjük ki.

$$\frac{\Gamma\,\mathsf{wf}}{\Gamma \vdash \mathsf{error} : \tau} \tag{5.38}$$

Az operációs szemantika a következő szabállyal egészül ki.

$$\frac{}{\mathsf{error}\,\mathsf{err}} \tag{5.39}$$

5.25. Feladat. *Egészítsük ki az 5.22. tételt hibákkal. Tehát bizonyítsuk be, hogy ha $e : \tau$, akkor vagy $e\,\mathsf{err}$, vagy $e\,\mathsf{val}$, vagy létezik olyan e', hogy $e \longmapsto e'$.*

5.26. Feladat. *Egészítsük ki a nyelvet (szintaxist, típusrendszert, operációs szemantikát) egy operátorral, mely egy szövegnek kiveszi az első betűjét (és egy egy karakter hosszú szöveget készít belőle). Ha üres szövegre alkalmazzuk, hibát adjon.*

6. Függvények

6.1. Elsőrendű függvények

A számok és szövegek nyelvének típusait (5.1) lecseréljük a következőre.

$$\rho, \rho_1, ... \in \mathsf{BTy} ::= \mathsf{str} \,|\, \mathsf{int} \tag{6.1}$$

$$\tau, \tau', ... \in \mathsf{Ty} \quad ::= \rho \,|\, \rho_1 \to \rho_2 \tag{6.2}$$

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

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

A kifejezéseket kiegészítjük függvénnyel és függvény-alkalmazással.

$$e, e', ... \in \mathsf{Exp} ::= ... \,|\, \mathsf{fun}^\rho x.e \,|\, \mathsf{app}\, e\, e' \tag{6.3}$$

A fun operátor aritása $(\mathsf{BTy}, \mathsf{Exp}.\mathsf{Exp})\mathsf{Exp}$, második paraméterében köt egy Exp fajtájú kifejezést. Első paramétere (felső indexben) megadja a függvény értelmezési tartományát.

A típusrendszert a következő szabályokkal egészítjük ki.

$$\frac{\Gamma, x : \rho_1 \vdash e_2 : \rho_2}{\Gamma \vdash \mathsf{fun}^{\rho_1} x.e_2 : \rho_1 \to \rho_2} \tag{6.4}$$

$$\frac{\Gamma \vdash e : \rho_1 \to \rho_2 \qquad \Gamma \vdash e_1 : \rho_1}{\Gamma \vdash \mathsf{app}\; e\, e_1 : \rho_2} \tag{6.5}$$

6.2. Feladat. *Írjuk fel azt az $\mathsf{str} \to \mathsf{str}$ típusú függvényt, ami egy szöveget háromszor egymás után másol.*

Többparaméteres függvényt majd akkor tudunk megadni, ha bevezettük a szorzat típusokat (7. fejezet), ekkor pl. az összeadást elvégző függvény típusa $(\mathsf{int} \times \mathsf{int}) \to \mathsf{int}$ lesz.

Az 5.2–5.8. lemmák az elsőrendű függvényekkel kiegészített típusrendszerre is igazak, bizonyításuk ugyanolyan módon történik.

6.3. 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 $\mathsf{fun} x.x$ kifejezésnek.*

Az operációs szemantikát a következő szabályokkal egészítjük ki.

$$\frac{}{\mathsf{fun}^\rho x.e\,\mathsf{val}} \tag{6.6}$$

$$\frac{e \longmapsto e'}{\mathsf{app}\, e\, e_1 \longmapsto \mathsf{app}\, e'\, e_1} \tag{6.7}$$

$$\left[\frac{e\,\mathsf{val} \qquad e_1 \longmapsto e_1'}{\mathsf{app}\, e\, e_1 \longmapsto \mathsf{app}\, e\, e_1'}\right] \tag{6.8}$$

$$\frac{[e_1\,\mathsf{val}]}{\mathsf{app}\,(\mathsf{fun}^{\rho} x.e_2)\, e_1 \longmapsto e_2[x \mapsto e_1]} \tag{6.9}$$

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

A függvény típus bevezető operátora a fun, eliminációs operátora az app. A 6.7–6.8. szabályok sorrendi szabályok, míg a 6.9. 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 (5.18. lemma) és determinisztikus (5.19. lemma).

Továbbá igaz a típusmegőrzés (5.21) és a haladás (5.22) tétele is, utóbbi bizonyításához a kanonikus alakok lemmáját (5.23) ki kell egészítenünk azzal, hogy ha $e : \rho_1 \to \rho_2$ és $e\,\mathsf{val}$, akkor $e = \mathsf{fun}^{\rho_1} x.e'$ valamely x-re és e'-re, melyekre $x : \rho_1 \vdash e' : \rho_2$.

6.2. Magasabbrendű függvények

TODO: Curry-Uncurry-ről beszélni.

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

$$\tau, \tau', \ldots \in \mathsf{Ty} ::= \mathsf{str} \,|\, \mathsf{int} \,|\, \tau_1 \to \tau_2 \tag{6.10}$$

6.4. Feladat. *Írjunk fel 10 különböző τ-t.*

Kifejezések:

$$e, e', \ldots \in \mathsf{Exp} ::= \ldots \,|\, \lambda^{\tau} x.e \,|\, e\, e' \tag{6.11}$$

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

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

$$\frac{\Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \lambda^{\tau_1} x.e_2 : \tau_1 \to \tau_2} \tag{6.12}$$

$$\frac{\Gamma \vdash e : \tau_1 \to \tau_2 \qquad \Gamma \vdash e_1 : \tau_1}{\Gamma \vdash e\, e_1 : \tau_2} \tag{6.13}$$

Most már tudunk többparaméteres függvényeket is megadni. Például egy $\mathsf{int} \to (\mathsf{str} \to \mathsf{int})$ 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 $\to$ operátor jobbra zárójeleződik, tehát $\tau_1 \to \tau_2 \to \tau_3 = \tau_1 \to (\tau_2 \to \tau_3)$.

6.5. Feladat. *Adjunk meg* $\mathsf{int} \to (\mathsf{str} \to \mathsf{int})$ *típusú függvényt, mely kezd valamit az* str *típusú paraméterével is.*

6.6. Feladat. *Adjuk meg az* $\mathsf{apply}_3 : (\mathsf{str} \to \mathsf{str}) \to \mathsf{str} \to \mathsf{str}$ *függvényt, mely az első paramétereként kapott függvényt háromszor alkalmazza a második paraméterében kapott szövegre.*

Az 5.2–5.8. lemmák a magasabbrendű függvényekkel kiegészített típusrendszerre is igazak. Pl. inverzió (5.4 lemma).

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

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

Az operációs szemantika az elsőrendű esettel analóg.

$$\frac{}{\lambda^\tau x.e\,\mathsf{val}} \tag{6.14}$$

$$\frac{e \longmapsto e'}{e\, e_1 \longmapsto e'\, e_1} \tag{6.15}$$

$$\left[\frac{e\,\mathsf{val} \qquad e_1 \longmapsto e_1'}{e\, e_1 \longmapsto e\, e_1'}\right] \tag{6.16}$$

$$\frac{[e_1\,\mathsf{val}]}{(\lambda^\tau x.e_2)\, e_1 \longmapsto e_2[x \mapsto e_1]} \tag{6.17}$$

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 (5.18. lemma) és determinisztikus (5.19. lemma).

Továbbá igaz a típusmegőrzés (5.21) és a haladás (5.22) tétele is, utóbbi bizonyításához a kanonikus alakok lemmáját (5.23) ki kell egészítenünk azzal, hogy ha $e : \tau_1 \to \tau_2$ és $e\,\mathsf{val}$, akkor $e = \lambda^{\tau_1} x.e'$ valamely x-re és e'-re, melyekre $x : \tau_1 \vdash e' : \tau_2$.

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

6.9. Feladat. *Értékeld ki név és érték szerinti paraméterátadással a* $(\lambda^{\mathsf{int}} x.x + x)\,(1+1)$ *kifejezést.*

6.10. Feladat. *Értékeld ki a* $\Big((\lambda^{\mathsf{int}\to\mathsf{int}} f.\lambda^{\mathsf{int}} x.f\,(f\,x))\,(\lambda^{\mathsf{int}} x.x+1)\Big)\,(3+1)$ *kifejezést.*

7. Véges adattípusok

Harper könyv: IV. rész.

7.1. 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 $\langle e, e' \rangle$ pár érték, míg mohó szemantika esetén szükséges, hogy e és e' már eleve értékek legyenek.

7.1.1. Nulláris és bináris szorzat típus

A szintaxis a következő.

$$\tau, \tau', ... \in \mathsf{Ty} \;\; ::= ... \,|\, \top \,|\, \tau \times \tau' \tag{7.1}$$

$$e, e', ... \in \mathsf{Exp} ::= ... \,|\, \mathsf{tt} \,|\, \langle e_1, e_2 \rangle \,|\, \mathsf{proj}_1\, e \,|\, \mathsf{proj}_2\, e \tag{7.2}$$

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 (binary product) Descartes-szorzatnak vagy keresztszorzatnak is nevezik.

A típusrendszer a következő.

$$\frac{\Gamma\,\mathsf{wf}}{\Gamma \vdash \mathsf{tt} : \top} \tag{7.3}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2} \tag{7.4}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathsf{proj}_1\, e : \tau_1} \tag{7.5}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathsf{proj}_2\, e : \tau_2} \tag{7.6}$$

Az operációs szemantika a következő.

$$\frac{}{\mathsf{tt}\,\mathsf{val}} \tag{7.7}$$

$$\frac{[e_1\,\mathsf{val}] \qquad [e_2\,\mathsf{val}]}{\langle e_1, e_2 \rangle\,\mathsf{val}} \tag{7.8}$$

$$\left[\frac{e_1 \longmapsto e_1'}{\langle e_1, e_2 \rangle \longmapsto \langle e_1', e_2 \rangle}\right] \tag{7.9}$$

[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.

$$\left[\frac{e_1\ \mathsf{val} \qquad e_2 \longmapsto e_2'}{\langle e_1, e_2\rangle \longmapsto \langle e_1, e_2'\rangle}\right] \tag{7.10}$$

$$\frac{e \longmapsto e'}{\mathsf{proj}_1\, e \longmapsto \mathsf{proj}_1 e'} \tag{7.11}$$

$$\frac{e \longmapsto e'}{\mathsf{proj}_2\, e \longmapsto \mathsf{proj}_2 e'} \tag{7.12}$$

$$\frac{[e_1\ \mathsf{val}] \qquad [e_2\ \mathsf{val}]}{\mathsf{proj}_1\, \langle e_1, e_2\rangle \longmapsto e_1} \tag{7.13}$$

$$\frac{[e_1\ \mathsf{val}] \qquad [e_2\ \mathsf{val}]}{\mathsf{proj}_2\, \langle e_1, e_2\rangle \longmapsto e_2} \tag{7.14}$$

A szögletes zárójelbe tett feltételek és szabályok csak a mohó szemantikában használatosak. A $-,-$ operátor konstruktor, míg a proj_1 és proj_2 az eliminátorok. Mohó kiértékelés esetén az utasítás szabályok (7.13–7.14) csak akkor működnek, ha a $-,-$ operátor mindkét paramétere ki van értékelve.

7.1. Feladat. *Hány olyan kifejezés van, mely érték (mohó operációs szemantikában), és* $(\top \times \top) \times \top$ *típusú illetve* $\top \times (\top \times \top)$ *típusú?*

7.2. Feladat. *Értékeld ki a* $(\lambda^{\mathsf{int}} x.\langle x+3, x+4\rangle)\,(1+2)$ *kifejezést az alábbi négy módon:*

- *Érték szerint, mohón.*
- *Név szerint, mohón.*
- *Érték szerint, lustán.*
- *Név szerint, lustán.*

Az 5.2–5.8, 5.18, 5.19 lemmák, és az 5.21) és az 5.22. tételek mind igazak erre a típusrendszere is.

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

A szorzat típusokkal lehet megadni többparaméteres ill. több visszatérési értékű függvényeket.

7.4. Feladat. *Add meg a három-paraméteres összeadás függvényt, melynek típusa* $(\mathsf{int} \times \mathsf{int} \times \mathsf{int}) \to \mathsf{int}$.

7.1.2. Általános véges szorzat típusok

Van egy $I = \{i_1, ..., i_n\}$ véges halmazunk, mely neveket tartalmaz, és egy olyan szorzat típust adunk meg, mely minden névhez tartalmaz egy elemet.

A szintaxisban a típusok a következők.

$$\tau, \tau', ... \in \mathsf{Ty} ::= ... \mid \Pi(i_1 \hookrightarrow \tau_1, ..., i_n \hookrightarrow \tau_n) \tag{7.15}$$

Π aritása $(\mathsf{Ty}^I)\mathsf{Ty}$, ami azt jelenti, hogy I minden elemére meg kell adnunk egy típust (A^B-vel jelöljük a B-ből A-ba történő leképezések halmazát). A leképezés jelölésére használjuk a $\hookrightarrow$ jelölést, a megadás sorrendje nem számít, tehát pl. $(i \hookrightarrow a, j \hookrightarrow b)$ megegyezik $(j \hookrightarrow b, i \hookrightarrow a)$-val.

A kifejezések a következők.

$$e, e', ... \in \mathsf{Exp} ::= ... \mid \langle i_1 \hookrightarrow e_1, ..., i_n \hookrightarrow e_n\rangle \mid \mathsf{proj}_{i_k}\, e \tag{7.16}$$

$\langle - \rangle$ aritása $(\mathsf{Exp}^I)\mathsf{Exp}$. proj_i aritása $(\mathsf{Exp})\mathsf{Exp}$, minden $k \in \{1, ..., n\}$-re van egy ilyen operátorunk.

Például az $I = \{\mathsf{name}, \mathsf{age}, \mathsf{postcode}\}$ egy olyan rekord típust ad meg, mely egy nevet, életkort és irányítószámot tartalmaz.

$$\Pi\big(\mathsf{name} \hookrightarrow \mathsf{str}, \mathsf{age} \hookrightarrow \mathsf{int}, \mathsf{postcode} \hookrightarrow \mathsf{int}\big)$$

Egy eleme így adható meg.

$$\langle \mathsf{name} \hookrightarrow \text{"}\texttt{Judit}\text{"}, \mathsf{age} \hookrightarrow 38, \mathsf{postcode} \hookrightarrow 1092\rangle$$

A típusrendszer a következő.

$$\frac{\Gamma\,\mathsf{wf} \qquad \Gamma \vdash e_1 : \tau_1 \qquad ... \qquad \Gamma \vdash e_n : \tau_n}{\Gamma \vdash \langle i_1 \hookrightarrow e_1, ..., i_n \hookrightarrow e_n\rangle : \Pi(i_1 \hookrightarrow \tau_1, ..., i_n \hookrightarrow \tau_n)} \tag{7.17}$$

$$\frac{\Gamma \vdash e : \Pi(i_1 \hookrightarrow \tau_1, ..., i_n \hookrightarrow \tau_n)}{\Gamma \vdash \mathsf{proj}_{i_k}\, e : \tau_k} \tag{7.18}$$

$\Pi(i_1 \hookrightarrow \tau_1, ..., i_n \hookrightarrow \tau_n)$-nek ugyanannyi eleme van, mint a $\tau_1 \times ... \times \tau_n$ típusnak, csak előbbi elemeinek a megadása kényelmesebb: név szerint adjuk meg az egyes komponenseket, és a sorrend sem számít. Ezenkívül név szerint ki tudjuk őket vetíteni, nem kell iterálva alkalmaznunk a proj_1, proj_2 operátorokat.

Operációs szemantika.

$$\frac{[e_1\,\mathsf{val} \qquad ... \qquad e_n\,\mathsf{val}]}{\langle i_1 \hookrightarrow e_1, ..., i_n \hookrightarrow e_n\rangle\,\mathsf{val}} \tag{7.19}$$

$$\left[\frac{e_1\,\mathsf{val} \qquad ... \qquad e_{k-1}\,\mathsf{val} \qquad e_k \longmapsto e_k'}{\langle i_1 \hookrightarrow e_1, ..., i_n \hookrightarrow e_n\rangle \longmapsto \langle i_1 \hookrightarrow e_1, ..., i_{k-1} \hookrightarrow e_{k-1}, i_k \hookrightarrow e_k', i_{k+1} \hookrightarrow e_{k+1}, ..., i_n \hookrightarrow e_n\rangle}\right] \tag{7.20}$$

$$\frac{e \longmapsto e' \qquad k \in \{1, ..., n\}}{\mathsf{proj}_{i_k}\, e \longmapsto \mathsf{proj}_{i_k}\, e'} \tag{7.21}$$

$$\frac{[e_1\,\mathsf{val} \qquad ... \qquad e_n\,\mathsf{val}]}{\mathsf{proj}_{i_k}\, \langle i_1 \hookrightarrow e_1, ..., i_n \hookrightarrow e_n\rangle \longmapsto e_{i_k}} \tag{7.22}$$

A szögletes zárójellel megadott feltételek és szabály mohó kiértékelés esetén van megadva.

7.5. Feladat. *Jelöld be, hogy melyik a konstruktor, eliminátor, melyek a sorrendi és melyek az utasítás szabályok.*

7.6. Feladat. *A nulláris és a bináris szorzat típusokat megkapjuk, ha I az üres halmaz ill. a kételemű halmaz. Mutasd meg, hogy az így megadott nulláris és bináris típus ugyanúgy viselkedik típusrendszer és operációs szemantika szempontjából, mint a 7.1.1. alfejezetben megadott.*

7.7. Feladat. *Bizonyítsd be a típusmegmaradás és a haladás tételét.*

7.2. Összeg típusok

7.2.1. Nulláris és bináris összeg

Szintaxis.

$$\tau, \tau', ... \in \mathsf{Ty} \;::=\; ... \,|\, \bot \,|\, \tau_1 + \tau_2 \tag{7.23}$$

$$e, e', ... \in \mathsf{Exp} \;::=\; ... \,|\, \mathsf{abort}^\tau\, e \,|\, \mathsf{inj_1}^{\tau_1,\tau_2}\, e \,|\, \mathsf{inj_2}^{\tau_1,\tau_2}\, e \,|\, \mathsf{case}\, e\, x_1.e_1\, x_2.e_2 \tag{7.24}$$

A nulláris összeg típus (bottom, 0 típus, void típus) egy olyan választási lehetőséget ad meg, ahol egy alternatíva sincs. Emiatt nincs konstruktora. Az eliminációs operátora pedig abortálja a számítást, amint kap egy bottom típusú értéket (ami nem lehetséges). A bináris összeg típusba kétféleképpen injektálhatunk: vagy az első, vagy a második komponensébe, ezt jelöli $\mathsf{inj_1}$ és $\mathsf{inj_2}$. Ezek aritása $(\mathsf{Ty}, \mathsf{Ty}, \mathsf{Exp})\mathsf{Exp}$. Egy összeg típusú kifejezést esetszétválasztással (case) tudunk eliminálni, aritása $(\mathsf{Exp}, \mathsf{Exp}.\mathsf{Exp}, \mathsf{Exp}.\mathsf{Exp})\mathsf{Exp}$.

Típusrendszer.

$$\frac{\Gamma \vdash e : \bot}{\Gamma \vdash \mathsf{abort}^\tau\, e : \tau} \tag{7.25}$$

$$\frac{\Gamma \vdash e_1 : \tau_1}{\Gamma \vdash \mathsf{inj_1}^{\tau_1,\tau_2}\, e_1 : \tau_1 + \tau_2} \tag{7.26}$$

$$\frac{\Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \mathsf{inj_2}^{\tau_1,\tau_2}\, e_2 : \tau_1 + \tau_2} \tag{7.27}$$

$$\frac{\Gamma \vdash e : \tau_1 + \tau_2 \qquad \Gamma, x_1 : \tau_1 \vdash e_1 : \tau \qquad \Gamma, x_2 : \tau_2 \vdash e_2 : \tau}{\Gamma \vdash \mathsf{case}\, e\, x_1.e_1\, x_2.e_2 : \tau} \tag{7.28}$$

Operációs szemantika. Az $\mathsf{inj_1}$ és $\mathsf{inj_2}$ típusparamétereit nem írtuk ki a rövidség kedvéért, de ott vannak.

$$\frac{e \longmapsto e'}{\mathsf{abort}^\tau\, e \longmapsto \mathsf{abort}^\tau\, e'} \tag{7.29}$$

$$\frac{[e\, \mathsf{val}]}{\mathsf{inj_1}\, e\, \mathsf{val}} \tag{7.30}$$

$$\frac{[e\, \mathsf{val}]}{\mathsf{inj_2}\, e\, \mathsf{val}} \tag{7.31}$$

$$\left[\frac{e \longmapsto e'}{\mathsf{inj_1}\, e \longmapsto \mathsf{inj_1}\, e'}\right] \tag{7.32}$$

$$\left[\frac{e \longmapsto e'}{\mathsf{inj_2}\, e \longmapsto \mathsf{inj_2}\, e'}\right] \tag{7.33}$$

$$\frac{e \longmapsto e'}{\mathsf{case}\, e\, x_1.e_1\, x_2.e_2 \longmapsto \mathsf{case}\, e'\, x_1.e_1\, x_2.e_2} \tag{7.34}$$

$$\frac{[e\, \mathsf{val}]}{\mathsf{case}\, (\mathsf{inj_1}\, e)\, x_1.e_1\, x_2.e_2 \longmapsto e_1[x_1 \mapsto e]} \tag{7.35}$$

$$\frac{[e\, \mathsf{val}]}{\mathsf{case}\, (\mathsf{inj_2}\, e)\, x_1.e_1\, x_2.e_2 \longmapsto e_2[x_2 \mapsto e]} \tag{7.36}$$

7.8. Feladat. *Add meg a* Bool *típust összeg típusként, és az if-then-else konstrukciót.*

7.9. Feladat. *Van -e különbség a lusta és a mohó if-then-else között?*

7.10. Feladat. *Add meg a tagadás, logika és, logikai vagy függvényeket a* case *operátor használatával.*

Az 5.2–5.8, 5.18, 5.19 lemmák, és az 5.21) és az 5.22. tételek mind igazak erre a típusrendszere is.

7.11. Feladat. *Bizonyítsd be a típusozás unicitását! Mi lenne, ha nem adnánk meg a típusokat az injektáló operátoroknál?*

7.2.2. Általános véges összeg típusok

$I = \{i_1, ..., i_n\}$ véges halmaz. Szintaxis.

$$\tau, \tau', ... \in \mathsf{Ty} \ ::= ... \,|\, \Sigma(i_1 \hookrightarrow \tau_1, ..., i_n \hookrightarrow \tau_n) \tag{7.37}$$

$$e, e', ... \in \mathsf{Exp} ::= ... \,|\, \mathsf{inj}_{i_k}{}^{i_1\hookrightarrow\tau_1,...,i_n\hookrightarrow\tau_n}\, e \,|\, \mathsf{case}\, e\, (i_1 \hookrightarrow x_1.e_1, ..., i_n \hookrightarrow x_n.e_n) \tag{7.38}$$

Σ aritása $(\mathsf{Ty}^I)\mathsf{Ty}$. Itt sem számít a sorrend. inj_{i_k} aritása $(\mathsf{Ty}^I, \mathsf{Exp})\mathsf{Exp}$. Az általános case aritása $(\mathsf{Exp}, (\mathsf{Exp}.\mathsf{Exp})^I)\mathsf{Exp}$.

Típusrendszer.

$$\frac{\Gamma \vdash e : \tau_k}{\Gamma \vdash \mathsf{inj}_{i_k}{}^{i_1\hookrightarrow\tau_1,...,i_n\hookrightarrow\tau_n}\, e : \Sigma(i_1 \hookrightarrow \tau_1, ..., i_n \hookrightarrow \tau_n)} \tag{7.39}$$

$$\frac{\Gamma \vdash e : \Sigma(i_1 \hookrightarrow \tau_1, ..., i_n \hookrightarrow \tau_n) \qquad \Gamma, x_1 : \tau_1 \vdash e_1 : \tau \qquad ... \qquad \Gamma, x_n : \tau_n \vdash e_n : \tau}{\Gamma \vdash \mathsf{case}\, e\, x_1.e_1 \,...\, x_n.e_n : \tau} \tag{7.40}$$

7.12. Feladat. *Írd fel az operációs szemantikát, a lusta és a mohó változatot is!*

7.13. Feladat. *Mutasd meg, hogyan lesz a nulláris és a bináris összeg speciális esete az általánosnak!*

7.3. Véges típusok alkalmazásai

Fontos megkülönböztetni a $\top$ és a $\bot$ típusokat. A $\top$-nak pontosan egy eleme van (ezt úgy értjük, hogy egy olyan kifejezés van, mely érték, és típusa $\top$), és nem hordoz semmi információt. Ezt unitnak nevezik a legtöbb nyelvben, de sokszor hibásan voidnak (üresnek) nevezik. Utóbbi esetben arra gondolnak, hogy nincs benne információ, nem pedig arra, hogy nincs egy eleme sem. A $\bot$-nak nincs egy eleme sem, ez a teljes információ: ha van egy $\bot$ típusú értékünk (ami lehetetlen), akkor tetszőleges más típusú értéket létre tudunk hozni (abort művelet).

Bool típus.

Felsorolások, pl. Észak, Dél, Kelet, Nyugat. Ez tkp. $1 + 1 + 1 + 1$ típus. Ilyen a beépített karakter típus is, általában egyenlőség-vizsgálattal van megadva a case ezekre.

Opt típus, mely opcionálisan tartalmaz valamilyen értéket. Szintaxis.

$$\tau, \tau', \ldots \in \mathsf{Ty} \quad ::= \ldots \mid \mathsf{Opt}\,\tau \tag{7.41}$$

$$e, e', \ldots \in \mathsf{Exp} ::= \ldots \mid \mathsf{null}^{\tau} \mid \mathsf{just}\,e \mid \mathsf{ifnull}\,e\,e_1\,x.e_2 \tag{7.42}$$

Típusrendszer. Úgy tudunk eliminálni Opt-ból, ha megadjuk, mi történjen, ha null értéket kaptunk. Ezt fejezi ki az ifnull.

$$\frac{\Gamma\,\mathsf{wf}}{\Gamma \vdash \mathsf{null}^{\tau} : \mathsf{Opt}\,\tau} \tag{7.43}$$

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathsf{just}\,e : \mathsf{Opt}\,\tau} \tag{7.44}$$

$$\frac{\Gamma \vdash e : \mathsf{Opt}\,\tau \qquad \Gamma \vdash e_1 : \tau' \qquad \Gamma, x : \tau \vdash e_2 : \tau'}{\Gamma \vdash \mathsf{ifnull}\,e\,e_1\,x.e_2 : \tau'} \tag{7.45}$$

Operációs szemantika.

$$\frac{}{\mathsf{null}^{\tau}\,\mathsf{val}} \tag{7.46}$$

$$\frac{[e\,\mathsf{val}]}{\mathsf{just}\,e\,\mathsf{val}} \tag{7.47}$$

$$\left[\frac{e \longmapsto e'}{\mathsf{just}\,e \longmapsto \mathsf{just}\,e'}\right] \tag{7.48}$$

$$\frac{e \longmapsto e'}{\mathsf{ifnull}\,e\,e_1\,x.e_2 \longmapsto \mathsf{ifnull}\,e'\,e_1\,x.e_2} \tag{7.49}$$

$$\frac{}{\mathsf{ifnull}\,\mathsf{null}^{\tau}\,e_1\,x.e_2 \longmapsto e_1} \tag{7.50}$$

$$\frac{[e\,\mathsf{val}]}{\mathsf{ifnull}\,(\mathsf{just}\,e)\,e_1\,x.e_2 \longmapsto e_2[x \mapsto e]} \tag{7.51}$$

7.14. Feladat. *Mutasd meg, hogy ha véges összegként adjuk meg az* Opt *típust, a fenti típusrendszer és operációs szemantika szabályait megkapjuk.*

7.15. Feladat. *Hogyan tudunk objektum-orientált programozási nyelveken (pl. Java) megadni összeg típusokat?*

τ_1 és τ_2 típusokat *izomorfnak* nevezzük, ha megadhatók f_1 és f_2 kifejezések, melyekre $x_1 : \tau_1 \vdash f_2 : \tau_2$ és $x_2 : \tau_2 \vdash f_1 : \tau_1$ és bármely e_1 τ_1 típusú értékre igaz, hogy $f_1[x_2 \mapsto (f_2[x_1 \mapsto e_1])] \longmapsto^* e_1$ és fordítva, bármely e_2 τ_2 típusú értékre igaz, hogy $f_2[x_1 \mapsto (f_1[x_2 \mapsto e_2])] \longmapsto^* e_2$.

7.16. Feladat. *Döntsd el, hogy az alábbi típusok izomorfak -e, ha igen, add meg f_1-et és f_2-t!*

$$\begin{array}{ll}
\mathsf{Bool} \times \top & \textit{és } \mathsf{Bool} \\
(\top + \top) + \top & \textit{és } \top + (\top + \top) \\
(\tau + \tau') \times \tau'' & \textit{és } (\tau \times \tau'') + (\tau' \times \tau'') \\
\tau + \bot & \textit{és } \tau \\
\tau \times \tau' & \textit{és } \tau' \times \tau \\
\tau \times \bot & \textit{és } \bot
\end{array}$$

7.17. Feladat. *Mutasd meg, hogy a véges típusok a fenti izomorfizmust egyenlőségnek véve kommutatív félgyűrűt (semiring)-et alkotnak.*

8. Konstruktív ítéletlogika

Harper könyv: XI. rész.

Ebben a fejezetben a konstruktív ítéletlogikával (propozícionális, nulladrendű) logikával és az eddig tanult típusrenszerekkel való kapcsolatával foglalkozunk.

8.1. Szintaxis

$$A, B, ... \in \mathsf{Prop} ::= X \mid \top \mid \bot \mid A \wedge B \mid A \vee B \mid A \supset B \tag{8.1}$$

Egy konstruktív ítéletlogikai állítás (propozíció) vagy egy ítéletváltozó ($X, Y, ...$-al jelöljük), vagy igaz, vagy hamis, vagy egy konjunkció, diszjunkció vagy egy implikáció.

8.2. Bizonyítások

Bevezetünk egy ítéletet, mely $A_1, ..., A_n \vdash A$ alakú, mely azt mondja, hogy A_1, ..., A_n-t feltéve A bizonyítható. A feltételek (hipotézisek) listáját Δ-val jelöljük és a következőképp adjuk meg.

$$\Delta, \Delta_1 ... \in \mathsf{Hipos} ::= \cdot \mid \Delta, A \tag{8.2}$$

Először megadunk egy újabb ítéletet, mely azt mondja, hogy a Δ feltételek között szerepel az A állítás: $\Delta \ni A$.

$$\frac{}{\Delta, A \ni A} \tag{8.3}$$

$$\frac{\Delta \ni A}{\Delta, B \ni A} \tag{8.4}$$

Megadjuk a bizonyíthatóság levezetési szabályait. Először is, ha feltettünk valamit, akkor azt bizonyíthatjuk.

$$\frac{\Delta \ni A}{\Delta \vdash A} \tag{8.5}$$

A logikai összekötők levezetési szabályait bevezető és eliminációs szabályokként adjuk meg.

Az igaz állítást bármikor levezethetjük, eliminációs szabálya nincs.

$$\frac{}{\Delta \vdash \top} \tag{8.6}$$

A hamis állításból bármi következik.

$$\frac{\Delta \vdash \bot}{\Delta \vdash A} \tag{8.7}$$

Konjunkciót akkor vezethetünk le, ha mindkét tagja igaz. Két eliminációs szabálya van, mellyel a benne levő információ nyerhető ki.

$$\frac{\Delta \vdash A \qquad \Delta \vdash B}{\Delta \vdash A \wedge B} \tag{8.8}$$

$$\frac{\Delta \vdash A \wedge B}{\Delta \vdash A} \tag{8.9}$$

$$\frac{\Delta \vdash A \wedge B}{\Delta \vdash B} \tag{8.10}$$

Diszjunkciót akkor vezethetünk le, ha vagy az egyik, vagy a másik komponense igaz. Eliminálni akkor tudunk $A \vee B$-ből C-be, ha A-t ill. B-t feltételezve is bizonyítható C.

$$\frac{\Delta \vdash A}{\Delta \vdash A \vee B} \tag{8.11}$$

$$\frac{\Delta \vdash B}{\Delta \vdash A \vee B} \tag{8.12}$$

$$\frac{\Delta \vdash A \vee B \qquad \Delta, A \vdash C \qquad \Delta, B \vdash C}{\Delta \vdash C} \tag{8.13}$$

Az $A \supset B$ implikáció bevezetéséhez B-t kell bizonyítanunk A feltételezésével. Az eliminációs szabály a modus ponens.

$$\frac{\Delta, A \vdash B}{\Delta \vdash A \supset B} \tag{8.14}$$

$$\frac{\Delta \vdash A \supset B \qquad \Delta \vdash A}{\Delta \vdash B} \tag{8.15}$$

A logikai tagadást rövidítésként adjuk meg, nem kell külön logikai összekötőt bevezetnünk hozzá: $\neg A := A \supset \bot$, vagyis A hamis, ha a hamisat implikálja.

Az „akkor és csak akkor” logikai összekötő a következő rövidítésként adható meg: $A \leftrightarrow B := (A \supset B) \wedge (B \supset A)$.

8.1. Feladat. *Bizonyítsuk be az alábbi állításokat (üres feltétellista mellett).*

1. $\big(X \supset (Y \supset Z)\big) \supset \big(Y \supset X \supset Z\big)$
2. $X \leftrightarrow (X \wedge \top)$
3. $X \leftrightarrow (X \vee \bot)$
4. $\big(X \supset (Y \wedge Z)\big) \supset \big((X \supset Y) \wedge (X \supset Z)\big)$
5. $\big((X \vee Y) \supset Z\big) \supset \big((X \supset Z) \wedge (Y \supset Z)\big)$
6. $X \supset \neg\neg X$
7. $\neg\neg(X \vee \neg X)$
8. $\neg\neg(\neg\neg X \supset X)$
9. $\neg\neg\neg X \leftrightarrow \neg X$
10. $\neg(X \leftrightarrow \neg X)$
11. $(\neg X \vee Y) \supset (X \supset Y)$
12. $\neg(X \vee Y) \leftrightarrow (\neg X \wedge \neg Y)$
13. $(\neg X \vee \neg Y) \supset \neg(X \wedge Y)$

8.3. Állítások mint típusok

A fenti levezetési szabályok eléggé hasonlítanak a véges típusok (nulláris és bináris szorzat és összeg) és a magasabbrendű függvények típusrendszeréhez. Az állítások helyett ott típusok voltak, mégpedig az alábbi táblázat szerint.

Prop	Ty
$\top$	$\top$
$\bot$	$\bot$
$A_1 \wedge A_2$	$\tau_1 \times \tau_2$
$A_1 \vee A_2$	$\tau_1 + \tau_2$
$A_1 \supset A_2$	$\tau_1 \to \tau_2$
X	lásd 11. fejezet

A levezetési szabályok megfeleltethetők egymásnak azzal a különbséggel, hogy a kifejezéseket az ítéletlogikában elhagyjuk. Például a modus ponens szabály kétféle változata.

$$\frac{\Delta \vdash A \supset B \qquad \Delta \vdash A}{\Delta \vdash B} \qquad \frac{\Gamma \vdash e : \tau_1 \to \tau_2 \qquad \Gamma \vdash e_1 : \tau_1}{\Gamma \vdash e\, e_1 : \tau_2}$$

A különbség annyi, hogy elhagyjuk a kifejezéseket: a bizonyításoknak nem adunk nevet, mert logikában nem érdekes, hogy pontosan melyik a bizonyítás.

8.2. Feladat. *A 8.1. feladatban megadott állításokat írjuk át típusokká. Az ítéletváltozók helyett használjunk típus-metaváltozókat, pl. τ_1, τ_2. Írjunk kifejezéseket, melyek az üres környezetben a megadott típusúak (bárhogyan is választjuk meg a típus-metaváltozókat).*

További információ:

- A bizonyításoknak is lehet operációs szemantikája, cut elimination-nek nevezzük. Ez pontosan megfelel a magasabbrendű függvények operációs szemantikájának (6.2).

8.4. Klasszikus logika

A fenti logika egy denotációs szemantikáját meg tudjuk adni igazságtáblával. Egy olyan függvényt, mely az ítéletváltozók halmazáról a $\{\mathsf{t}, \mathsf{f}\}$ halmazba képez, interpretációnak nevezzük. Egy A állítás jelenetését I interpretáció esetén az

$|A|^I \in \{\mathsf{t}, \mathsf{f}\}$ adja meg, $|-|^-$ a következőképp van definiálva.

$$
\begin{aligned}
|X|^I &= I(X) \\
|\top|^I &= \mathsf{t} \\
|\bot|^I &= \mathsf{f} \\
|A \wedge B|^I &= \begin{cases} \mathsf{t}, & \text{ha } |A|^I = \mathsf{t} \text{ és } |B|^I = \mathsf{t} \\ \mathsf{f}, & \text{egyébként} \end{cases} \\
|A \vee B|^I &= \begin{cases} \mathsf{t}, & \text{ha } |A|^I = \mathsf{t} \text{ vagy } |B|^I = \mathsf{t} \\ \mathsf{f}, & \text{egyébként} \end{cases} \\
|A \supset B|^I &= \begin{cases} \mathsf{t}, & \text{ha } |A|^I = \mathsf{f} \text{ vagy } |B|^I = \mathsf{t} \\ \mathsf{f}, & \text{egyébként} \end{cases}
\end{aligned}
$$

Hasonlóképp megadjuk egy feltétellista jelentését.

$$
\begin{aligned}
|\cdot|^I &= \mathsf{t} \\
|\Delta, A|^I &= \begin{cases} \mathsf{t}, & \text{ha } |\Delta|^I = \mathsf{t} \text{ és } |A|^I = \mathsf{t} \\ \mathsf{f}, & \text{egyébként} \end{cases}
\end{aligned}
$$

Azt mondjuk, hogy $I \models A$, ha $|A|^I = \mathsf{t}$. Azt mondjuk, hogy $\Delta \models A$, ha minden I-re, ha $|\Delta|^I = \mathsf{t}$, akkor $|A|^I = \mathsf{t}$.

A bizonyíthatóság és a fenti szemantika kapcsolata az, hogy $\Delta \vdash A$-ból következik, hogy $\Delta \models A$.

8.3. Feladat. *Bizonyítsd be!*

A másik irány (teljesség) csak akkor igaz, ha a logikát klasszikussá tesszük, vagyis a bizonyítások levezetési szabályaihoz hozzávesszük a kizárt harmadik elvének (tertium non datur) alábbi szabályát.

$$\frac{}{\Delta \vdash A \vee \neg A} \tag{8.16}$$

További információ:

– A konstruktív logika denotációs szemantikáját Kripke-modellekkel adhatjuk meg, ezek nem validálják a kizárt harmadik elvét. Megadható egy olyan Kripke modell, mely teljes, tehát $\Delta \models A$-ból következik $\Delta \vdash A$ a kizárt harmadik elvétől függetlenül.

– A kizárt harmadik elvének típusrendszerbeli megfelelője másfajta operációs szemantikát kíván, amit itt nem tárgyalunk (lásd Harper könyv 13. fejezet). Tehát a magasabbrendű függvények és véges típusok operációs szemantikája már nem megfelelő.

8.4. Feladat. *Bizonyítsuk be az alábbi állításokat (üres feltétellista mellett), a kizárt harmadik elvét is felhasználva.*

1. $(X \supset Y) \supset (\neg X \vee Y)$

2. $\neg(X \wedge Y) \supset (\neg X \vee \neg Y)$

9. Természetes számok

Harper könyv: III. rész, 9. fejezet.

Ebben a fejezetben bevezetjük a természetes számok típusát. A számok és szövegek típusrendszerében (**??**. fejezet) megadott int típusnak ad-hoc módon adtuk meg az operátorait. Most egy általános mechanizmust adunk meg, a *primitív rekurziót*, mellyel sokféle, természetes számokon értelmezett függvényt tudunk megadni. Ezt a típusrendszert Gödel System T-jének is nevezik.

9.1. Szintaxis

$$\begin{aligned}
\tau, \tau', ... \in \mathsf{Ty} \;\; &::= \mathsf{Nat} \,|\, \tau_1 \to \tau_2 \\
e, e', ... \in \mathsf{Exp} \;\; &::= x \,|\, \mathsf{zero} \,|\, \mathsf{suc}\, e \,|\, \mathsf{rec}\, e_0\; x.e_1\; e \,|\, \lambda^\tau x.e \,|\, e\, e'
\end{aligned}$$

Kétféle természetes szám van (két konstruktora van a természetes számoknak): zero (nulla) és $\mathsf{suc}\, e$, amely valamely más természetes számnak, e-nek a rákövetkezője. Pl. az 1-et úgy írjuk, hogy $\mathsf{suc}\,\mathsf{zero}$, a 2-t úgy, hogy $\mathsf{suc}\,(\mathsf{suc}\,\mathsf{zero})$ stb. A rec operátor a természetes számok eliminátora, aritása $(\mathsf{Exp}, \mathsf{Exp}.\mathsf{Exp}, \mathsf{Exp})\mathsf{Exp}$. Ezzel tudunk függvényeket megadni a természetes számokon. $\mathsf{rec}\, e_0\; x.e_1\; e$ az e_0 és $x.e_1$ által megadott függvény alkalmazva e bemenetre. Ha a bemenet nulla, a függvény e_0-t fog adni, ha a függvény kimenete valamilyen számra x, akkor a rákövetkezőjére e_1 (ami ugye függhet x-től).

9.2. Típusrendszer

A környezetre és változókra vonatkozó szabályok a szokásosak (5.4–5.9).

$$\frac{\Gamma\, \mathsf{wf}}{\Gamma \vdash \mathsf{zero} : \mathsf{Nat}} \tag{9.1}$$

$$\frac{\Gamma \vdash e : \mathsf{Nat}}{\Gamma \vdash \mathsf{suc}\, e : \mathsf{Nat}} \tag{9.2}$$

$$\frac{\Gamma \vdash e_0 : \tau \qquad \Gamma, x : \tau \vdash e_1 : \tau \qquad \Gamma \vdash e : \mathsf{Nat}}{\Gamma \vdash \mathsf{rec}\, e_0\; x.e_1\; e : \tau} \tag{9.3}$$

A függvényekre vonatkozó szabályok a szokásosak.

$$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda^{\tau_1} x.e : \tau_1 \to \tau_2} \tag{9.4}$$

$$\frac{\Gamma \vdash e : \tau_1 \to \tau_2 \qquad \Gamma \vdash e_1 : \tau_1}{\Gamma \vdash e\, e_1 : \tau_2} \tag{9.5}$$

9.3. Operációs szemantika

Értékek a nulla, a rákövetkező (mely egy érték rákövetkezője kell, hogy legyen mohó kiértékelésnél) és az absztrakció.

$$\frac{}{\mathsf{zero}\; \mathsf{val}} \tag{9.6}$$

$$\frac{[e\; \mathsf{val}]}{\mathsf{suc}\, e\; \mathsf{val}} \tag{9.7}$$

$$\overline{\lambda^{\tau} x.e\ \mathsf{val}} \tag{9.8}$$

$$\left[\frac{e \longmapsto e'}{\mathsf{suc}\, e \longmapsto \mathsf{suc}\, e'}\right] \tag{9.9}$$

$$\frac{e \longmapsto e'}{\mathsf{rec}\, e_0\, x.e_1\, e \longmapsto \mathsf{rec}\, e_0\, x.e_1\, e'} \tag{9.10}$$

$$\overline{\mathsf{rec}\, e_0\, x.e_1\ \mathsf{zero} \longmapsto e_0} \tag{9.11}$$

$$\frac{\mathsf{suc}\, e\ \mathsf{val}}{\mathsf{rec}\, e_0\, x.e_1\, (\mathsf{suc}\, e) \longmapsto e_1[x \mapsto \mathsf{rec}\, e_0\, x.e_1\, e]} \tag{9.12}$$

$$\frac{e \longmapsto e'}{e\, e_1 \longmapsto e'\, e_1} \tag{9.13}$$

$$\left[\frac{e\ \mathsf{val} \qquad e_1 \longmapsto e_1'}{e\, e_1 \longmapsto e\, e_1'}\right] \tag{9.14}$$

$$\frac{[e_1\ \mathsf{val}]}{(\lambda^{\tau} x.e_2)\, e_1 \longmapsto e_2[x \mapsto e_1]} \tag{9.15}$$

A szögletesen zárójelezett szabályok a mohó rákövetkezőhöz és érték szerinti függvényalkalmazáshoz kellenek. Ha nem vesszük őket hozzá az operációs szemantikához, akkor lusta rákövetkezőt és név szerinti paraméterátadást kapunk.

Például az összeadás függvényt az alábbi módon adjuk meg.

$$\lambda^{\mathsf{Nat}} y.\lambda^{\mathsf{Nat}} z.\mathsf{rec}\, z\, (x.\mathsf{suc}\, x)\, y : \mathsf{Nat} \to (\mathsf{Nat} \to \mathsf{Nat})$$

Ez a következőképp működik: y-t és z-ta akarjuk összeadni, emiatt y-on végzünk rekurziót. Ha $y = \mathsf{zero}$, akkor z-t adunk vissza (hiszen $0+z$ egyenlő z-vel). Ha $y = \mathsf{suc}\, e$, akkor x-ben megkötjük $\mathsf{rec}\, z\, (x.\mathsf{suc}\, x)\, e$ eredményét, majd hozzáadunk egyet. Pl. a következőképp zajlik az átírás mohó rákövetkező esetén, ha a bemenetek 2 és 1.

$$\begin{aligned}
&\Big(\big(\lambda^{\mathsf{Nat}} y.\lambda^{\mathsf{Nat}} z.\mathsf{rec}\, z\, (x.\mathsf{suc}\, x)\, y\big)\, (\mathsf{suc}\, (\mathsf{suc}\, \mathsf{zero}))\Big)\, (\mathsf{suc}\, \mathsf{zero}) \\
\overset{(9.13)}{\longmapsto}\ &\big(\lambda^{\mathsf{Nat}} z.\mathsf{rec}\, z\, (x.\mathsf{suc}\, x)\, (\mathsf{suc}\, (\mathsf{suc}\, \mathsf{zero}))\big)\, (\mathsf{suc}\, \mathsf{zero}) \\
\overset{(9.15)}{\longmapsto}\ &\mathsf{rec}\, (\mathsf{suc}\, \mathsf{zero})\, (x.\mathsf{suc}\, x)\, (\mathsf{suc}\, (\mathsf{suc}\, \mathsf{zero})) \\
\overset{(9.12)}{\longmapsto}\ &(\mathsf{suc}\, x)[x \mapsto \mathsf{rec}\, (\mathsf{suc}\, \mathsf{zero})\, (x.\mathsf{suc}\, x)\, (\mathsf{suc}\, \mathsf{zero})] = \mathsf{suc}\, \big(\mathsf{rec}\, (\mathsf{suc}\, \mathsf{zero})\, (x.\mathsf{suc}\, x)\, (\mathsf{suc}\, \mathsf{zero})\big) \\
\overset{(9.9)}{\longmapsto}\ &\mathsf{suc}\, \big((\mathsf{suc}\, x)[x \mapsto \mathsf{rec}\, (\mathsf{suc}\, \mathsf{zero})\, (x.\mathsf{suc}\, x)\, \mathsf{zero}]\big) = \mathsf{suc}\, \big(\mathsf{suc}\, (\mathsf{rec}\, (\mathsf{suc}\, \mathsf{zero})\, (x.\mathsf{suc}\, x)\, \mathsf{zero})\big) \\
\overset{(9.9)}{\longmapsto}\ &\mathsf{suc}\, \big(\mathsf{suc}\, (\mathsf{suc}\, \mathsf{zero})\big)
\end{aligned}$$

A nyilak fölé mindig csak az adott lépés levezetési fájának gyökerénél levő szabály sorszámát adtuk meg. Lusta kiértékelés esetén az utolsó két lépés elmarad, hiszen ekkor csak addig értékeljük ki az eredményt, hogy egy rákövetkezőnk van, nem foglalkozunk azzal, hogy a suc paraméterében is érték legyen.

A rekurzor primitív rekurziót valósít meg. A fenti definíciót rekurzívan ilyesmi szintaxissal is írhatnánk:

$$\begin{aligned}\mathsf{plus}\,y\,z := \mathsf{case}\,y\,\mathsf{of}&\\ &\mathsf{zero}\mapsto z\\ &\mathsf{suc}\,y'\mapsto \mathsf{suc}\,(\mathsf{plus}\,y'\,z)\end{aligned}$$

A rekurzió azért primitív, mert a plus függvény rekurzív hívása csak pontosan eggyel kisebb paraméteren lehetséges. Ha $y = \mathsf{suc}\,y'$ volt az első paraméter, akkor csak y'-n hívható meg a függvény.

$\overline{n}$-el jelöljük azt a kifejezést, ahol zero-ra n-szer van alkalmazva a suc operátor.

9.1. Feladat. *Add meg a pred függvényt, mely egy számhoz hozzárendeli a nála eggyel kisebb számot (0-hoz rendeljen 0-t).*

9.2. Feladat. *Add meg a természetes számok szorzását! (Egy olyan e kifejezést, mely $\mathsf{Nat}\to(\mathsf{Nat}\to\mathsf{Nat})$ típusú, és $(e\,\overline{n})\,\overline{n'}$ több lépésben $\overline{n*n'}$-re íródik át.)*

9.3. Feladat. *Bizonyítsd be, hogy ha az összeadás fenti definícióját e-vel jelöljük, akkor bármely $n\in\mathbb{N}$-re $e\,\overline{n}\,\mathsf{zero}\longmapsto^*\overline{n}$.*

9.4. Feladat. *Bizonyítsd a haladás és a típusmegmaradás tételét.*

9.4. Definiálható függvények

Azt mondjuk, hogy egy $f:\mathbb{N}\to\mathbb{N}$ függvény *definiálható* System T-ben, ha létezik olyan e_f kifejezés, mely $\mathsf{Nat}\to\mathsf{Nat}$ típusú, és igaz rá, hogy minden $n\in\mathbb{N}$-re

$$e_f\,\overline{n}\longmapsto^*\overline{f(n)}.$$

9.5. Feladat. *Mutassuk meg, hogy a duplázás függvény definiálható System T-ben!*

System T-ben (és az eddig tanult összes nyelvben) nem tudunk írni végtelen ciklust.

9.6. Tétel. *Ha $e:\tau$, akkor létezik egy olyan e', hogy $e'\,\mathsf{val}$ és $e\longmapsto^* e'$.*

Emiatt a System T-beli $\tau_1\to\tau_2$ típusú kifejezések úgy működnek, mint a matematikai függvények.

9.7. Tétel. *Létezik olyan matematikai függvény, mely nem definiálható System T-ben.*

A bizonyítás a (Cantor-féle) átlós metszés technikáját használja (hasonló bizonyítások: a természetes számok részhalmazai többen vannak, mint a természetes számok; Gödel-tétel; Turing-gépek megállásának eldönthetetlensége).

Bizonyítás. Először is létrehozunk egy kódolást, mely egy ABT-t egy számmal kódol. Ennek a részleteibe nem megyünk bele, de a lényeg, hogy minden e ABT-nek megfelel egy $\ulcorner e\urcorner$ természetes szám, és két különböző ABT-t különböző természetes számokkal reprezentálunk.

A következő matematikai függvényt adjuk meg. $u : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ tetszőleges $e : \mathsf{Nat} \to \mathsf{Nat}$ kifejezésre úgy van megadva, hogy $e\,\overline{m} \longmapsto^* \overline{u(\ulcorner e \urcorner)(m)}$. Tehát u egy olyan függvény, mely tetszőleges $e : \mathsf{Nat} \to \mathsf{Nat}$ program működését szimulálja. Mivel az átíró rendszer úgy van megadva, hogy mindig csak egy átírási lépést választhatunk, és mivel minden átírási sorozat véges, emiatt u jól definiált.

Tfh. hogy u definiálható System T-ben, mégpedig az $e_u : \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat}$ kifejezéssel, vagyis a következő igaz: $e_u\,\overline{m}\,\overline{n} \longmapsto^* \overline{u(m)(n)}$.

Ekkor megadjuk az $e_\Delta : \mathsf{Nat} \to \mathsf{Nat}$ kifejezést a következőképp: $e_\Delta := = \lambda^{\mathsf{Nat}} x.\mathsf{suc}\,(e_u\,x\,x)$. Ekkor egyrészt tudjuk, hogy u-ra teljesül, hogy

$$e_\Delta\,\overline{\ulcorner e_\Delta \urcorner} \longmapsto^* \overline{u(\ulcorner e_\Delta \urcorner)(\ulcorner e_\Delta \urcorner)},$$

másrészt e_Δ definíciójából és e_u tulajdonságából

$$e_\Delta\,\overline{\ulcorner e_\Delta \urcorner} \longmapsto \mathsf{suc}\,(e_u\,\overline{\ulcorner e_\Delta \urcorner}\,\overline{\ulcorner e_\Delta \urcorner}) \longmapsto^* \mathsf{suc}\left(\overline{u(\ulcorner e_\Delta \urcorner)(\ulcorner e_\Delta \urcorner)}\right).$$

Ebből az következik, hogy a kiértékelés nem egyértelmű, hiszen két különböző értéket is kapnánk ugyanabból a kifejezésből. Ez ellentmondás, tehát u nem definiálható System T-ben. □

10. Végtelen adattípusok

Harper könyv: V. rész.

10.1. Generikus programozás

Tfh. van egy $\mathsf{str} \to \mathsf{int}$ függvényünk, például ami megadja egy szöveg hosszát ($\lambda^{\mathsf{str}} x.|x|$). Ezt tudjuk alkalmazni egy str típusú kifejezésre. De hasonlóképp tudnánk alkalmazni $\mathsf{Bool} \times \mathsf{str}$ típusú kifejezésre is, úgy, hogy a Bool rész érintetlen marad, tehát $\langle b, s \rangle$-ből $\langle b, |s| \rangle$ lesz. Vagy egy $(\mathsf{int} + \mathsf{str}) \times \mathsf{str}$ kifejezésre is, ahol mindkét str-re alkalmaznánk, az int típusú értéket meg úgy hagynánk. Általánosságban: tetszőleges $\tau' \to \tau''$ függvényt szeretnénk kiterjeszteni egy α-t tartalmazó τ típusra, úgy, hogy egy $\tau[\alpha \mapsto \tau'] \to \tau[\alpha \mapsto \tau'']$ függvényt kapjunk. A generikus programozás ezt teszi lehetővé bizonyos τ típusok esetén. Ha τ-ban α-n kívül (nulláris és bináris) szorzat és összeg típusok lehetnek, akkor τ-t polinomiális típusoperátornak nevezzük. Ha függvény típus is lehet benne, akkor a pozitív típusoperátorok lesznek azok, melyekre a generikus művelet megadható.

10.1.1. Polinomiális típusoperátorok

A polinomiális típusoperátorok az algebrában tanult polinomokhoz hasonlóak, például

$$2 * \alpha^3 + \alpha^2 + 3 * \alpha + 1$$

helyett azt írjuk (valamilyen standard módon zárójelezve), hogy

$$(\top + \top) \times (\alpha \times \alpha \times \alpha) + \alpha \times \alpha + (\top + \top + \top) \times \alpha + \top.$$

Bevezetünk egy ítéletet, mely azt mondja, hogy egy α-t tartalmazó típus egy polinom, jelölése $\alpha.\tau\ \mathsf{poly}$. Levezetési szabályok:

$$\frac{}{\alpha.\alpha\ \mathsf{poly}} \tag{10.1}$$

$$\frac{}{\alpha.\top\ \mathsf{poly}} \tag{10.2}$$

$$\frac{\alpha.\tau_1\ \mathsf{poly} \qquad \alpha.\tau_2\ \mathsf{poly}}{\alpha.\tau_1 \times \tau_2\ \mathsf{poly}} \tag{10.3}$$

$$\frac{}{\alpha.\bot\ \mathsf{poly}} \tag{10.4}$$

$$\frac{\alpha.\tau_1\ \mathsf{poly} \qquad \alpha.\tau_2\ \mathsf{poly}}{\alpha.\tau_1 + \tau_2\ \mathsf{poly}} \tag{10.5}$$

10.1. Feladat. *Vezessük le, hogy* $\alpha.\bot + \big((\top \times \alpha) + \alpha \times \alpha\big)\ \mathsf{poly}$*!*

10.2. Feladat. *Mi lesz* $\Big(\bot + \big((\top \times \alpha) + \alpha \times \alpha\big)\Big)[\alpha \mapsto (\mathsf{int} + \mathsf{str})]$ *eredménye?*

A polinomiális típusoperátorok olyan adatstruktúrák, melyekben van egy lyuk (az α), ahová egy adott típust lehet helyettesíteni. Például az $\alpha.\alpha \times (\mathsf{Nat} + \alpha)$ az összes τ típusra megad egy $\tau \times (\mathsf{Nat} + \tau)$ típust.

A $\tau' \to \tau''$ függvény kiterjesztését a map operátor fogja megadni. Függvények helyett $x : \tau' \vdash e'' : \tau''$ kifejezést használunk.

$$e, e', \ldots \in \mathsf{Exp} ::= \ldots \mid \mathsf{map}^{\alpha.\tau}(x'.e'')\, e \tag{10.6}$$

$$\frac{\alpha.\tau\ \mathsf{poly} \qquad \Gamma, x' : \tau' \vdash e'' : \tau'' \qquad \Gamma \vdash e : \tau[\alpha \mapsto \tau']}{\Gamma \vdash \mathsf{map}^{\alpha.\tau}(x'.e'')\, e : \tau[\alpha \mapsto \tau'']} \tag{10.7}$$

Az e kifejezésben vannak τ' típusú részkifejezések, ezeket fogja a map operátor τ'' típusú részkifejezésekre cserélni. Az e'' művelet (mely τ'-ből τ''-be képez), mondja meg, hogy hogyan kell őket lecserélni. Az operációs szemantika azt fejezi ki, hogy ott kell végrehajtanunk az e'' műveletet, ahol α volt a polinom.

$$\frac{}{\mathsf{map}^{\alpha.\alpha}(x'.e'')\, e \longmapsto e''[x' \mapsto e]} \tag{10.8}$$

$$\frac{}{\mathsf{map}^{\alpha.\top}(x'.e'')\, e \longmapsto e} \tag{10.9}$$

$$\frac{}{\mathsf{map}^{\alpha.\tau_1 \times \tau_2}(x'.e'')\, e \longmapsto \langle \mathsf{map}^{\alpha.\tau_1}(x'.e'')\,(\mathsf{proj}_1\, e), \mathsf{map}^{\alpha.\tau_2}(x'.e'')\,(\mathsf{proj}_2\, e)\rangle} \tag{10.10}$$

$$\frac{}{\mathsf{map}^{\alpha.\bot}(x'.e'')\, e \longmapsto e} \tag{10.11}$$

$$\frac{}{\mathsf{map}^{\alpha.\tau_1 + \tau_2}(x'.e'')\, e \longmapsto \mathsf{case}\, e\, x_1.\big(\mathsf{inj}_1\,(\mathsf{map}^{\alpha.\tau_1}(x'.e'')\, x_1)\big)\, x_2.\big(\mathsf{inj}_2\,(\mathsf{map}^{\alpha.\tau_2}(x'.e'')\, x_2)\big)} \tag{10.12}$$

10.3. Feladat. *Mutasd meg, hogy bármely e kifejezésre*

$$\mathsf{map}^{\alpha.\top + (\mathsf{Bool} \times \alpha)}(x'.\mathsf{suc}\, x')\, \big(\mathsf{inj}_2\, \langle \mathsf{true}, e\rangle\big) \longmapsto^* \mathsf{inj}_2\, \langle \mathsf{true}, \mathsf{suc}\, e\rangle!$$

10.4. Feladat. *Mutasd meg, hogy ha* $\alpha.\tau\ \mathsf{poly}$ *és* $\alpha'.\tau'\ \mathsf{poly}$*, akkor* $\alpha.\tau'[\alpha' \mapsto \tau]\ \mathsf{poly}$.

10.5. Feladat. *Bizonyítsd be a típusmegmaradás tételét!*

10.6. Feladat. *Mutasd meg, hogy a konstans típusoperátorral végzett* map *nem csinál semmit. Vagyis, ha* $e\, \mathsf{val}$ *és* $e : \tau$*, akkor* $\mathsf{map}^{\alpha.\tau}(x'.e'')\, e \longmapsto^* e$ *függetlenül attól, hogy mi* e''*.*

10.1.2. Pozitív típusoperátorok

Szeretnénk függvénytípusokat is megengedni a típusoperátorokban. Ezeket megszorítjuk a szigorúan pozitív (strictly positive) típusoperátorokra: egy $\alpha.\tau_1 \to \tau_2$ típusoperátor szigorúan pozitív, ha (1) τ_1-ben nem szerepel α és (2) $\alpha.\tau_2$ is szigorúan pozitív. A pozitív, de nem szigorú típusoperátorokkal nem foglalkozunk.

Az $\alpha.\tau\ \mathsf{spos}$ ítélet azt fejezi ki, hogy $\alpha.\tau$ egy szigorúan pozitív típusoperátor. Levezetési szabályai:

$$\frac{}{\alpha.\alpha\ \mathsf{spos}} \tag{10.13}$$

$$\frac{}{\alpha.\top\ \mathsf{spos}} \tag{10.14}$$

$$\frac{\alpha.\tau_1\ \mathsf{spos} \qquad \alpha.\tau_2\ \mathsf{spos}}{\alpha.\tau_1 \times \tau_2\ \mathsf{spos}} \tag{10.15}$$

$$\frac{}{\alpha.\bot\ \mathsf{spos}} \tag{10.16}$$

$$\frac{\alpha.\tau_1\ \mathsf{poly} \qquad \alpha.\tau_2\ \mathsf{spos}}{\alpha.\tau_1 + \tau_2\ \mathsf{spos}} \tag{10.17}$$

$$\frac{\tau_1\ \mathsf{type} \qquad \alpha.\tau_2\ \mathsf{spos}}{\alpha.\tau_1 \to \tau_2\ \mathsf{spos}} \tag{10.18}$$

Az utolsó szabály az újdonság csak a polinomiális típusoperátorokhoz képest. $\tau_1\ \mathsf{type}$ azt jelenti, hogy τ_1 egy típus, nem szerepel benne típusváltozó, ennek levezetési szabályai:

$$\frac{}{\top\ \mathsf{type}} \tag{10.19}$$

$$\frac{}{\bot\ \mathsf{type}} \tag{10.20}$$

$$\frac{\tau_1\ \mathsf{type} \qquad \tau_2\ \mathsf{type}}{\tau_1 \times \tau_2\ \mathsf{type}} \tag{10.21}$$

$$\frac{\tau_1\ \mathsf{type} \qquad \tau_2\ \mathsf{type}}{\tau_1 + \tau_2\ \mathsf{type}} \tag{10.22}$$

$$\frac{\tau_1\ \mathsf{type} \qquad \tau_2\ \mathsf{type}}{\tau_1 \to \tau_2\ \mathsf{type}} \tag{10.23}$$

A map operátor típusozási szabálya.

$$\frac{\alpha.\tau\ \mathsf{spos} \qquad \Gamma, x' : \tau' \vdash e'' : \tau'' \qquad \Gamma \vdash e : \tau[\alpha \mapsto \tau']}{\Gamma \vdash \mathsf{map}^{\alpha.\tau}(x'.e'')\, e : \tau[\alpha \mapsto \tau'']} \tag{10.24}$$

Az operációs szemantika szabályai ugyanazok, mint a polinomiális típusoperátoroknál, és függvény típusra a következő szabályunk van.

$$\frac{}{\mathsf{map}^{\alpha.\tau_1 \to \tau_2}(x'.e'')\, e \longmapsto \lambda^{\tau_1} x_1.\mathsf{map}^{\alpha.\tau_2}(x'.e'')\,(e\, x_1)} \tag{10.25}$$

10.7. Feladat. *Egészítsd ki a 10.6. feladatot pozitív típusoperátorokra.*

További információ:

- A típusoperátorok funktorok.
- A map függvény megadható pozitív, de nem szigorúan pozitív típusoperátorokra is, ekkor foglalkoznunk kell a negatív típusoperátorokkal is.

10.2. Induktív és koinduktív típusok

Az induktív típusok elemei konstruktorok véges sokszor való egymásra alkalmazásai. Tehát, ha minden konstruktorra megadjuk, hogy ahhoz mit rendeljen egy függvény, azzal az induktív típus minden elemére megadtuk a függvényt. Ezt hívják rekurziónak (vagy indukciónak).

A koinduktív típusok elemei azok, melyeken véges sokszor lehet destrukciót végezni. Tehát, ha minden destruktorra meg van adva, hogy egy elem hogyan viselkedjen, azzal meg van határozva a koinduktív típus egy eleme. Ezt hívják generátornak (vagy koindukciónak).

Ebben a fejezetben általánosságban adjuk meg, hogy mi az, hogy (ko)induktív típus. A természetes számok pl. ennek egy speciális esete lesz.

10.2.1. Példák induktív és koinduktív típusokra

Természetes számok induktív típusát lásd a 9. fejezetben.

Bináris fák, leveleknél természetes számok induktív típusa. Induktív típusokat konstruktorokkal és egy rekurzorral adunk meg. Konstruktorok:

$$\frac{\Gamma \vdash e : \mathsf{Nat}}{\Gamma \vdash \mathsf{leaf}\, e : \mathsf{Tree}} \tag{10.26}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{Tree} \qquad \Gamma \vdash e_2 : \mathsf{Tree}}{\Gamma \vdash \mathsf{node}\, e_1\, e_2 : \mathsf{Tree}} \tag{10.27}$$

Rekurzor (eliminációs szabály):

$$\frac{\Gamma, x : \mathsf{Nat} \vdash e_1 : \tau \qquad \Gamma, x_1 : \tau, x_2 : \tau \vdash e_2 : \tau \qquad \Gamma \vdash e : \mathsf{Tree}}{\Gamma \vdash \mathsf{rec}'_{\mathsf{Tree}}\, x.e_1\, x_1.x_2.e_2\, e : \tau} \tag{10.28}$$

Lusta operációs szemantika: értékek a konstruktorok, egy sorrendi szabály van, mely a rekurzornak a Tree paraméterét értékeli ki, és két utasítás szabály, melyek megmondják, mit kell csinálni, ha a rekurzort egy konstruktorra alkalmaztunk.

$$\frac{}{\mathsf{leaf}\, e\, \mathsf{val}} \tag{10.29}$$

$$\frac{}{\mathsf{node}\, e_1\, e_2\, \mathsf{val}} \tag{10.30}$$

$$\frac{e \longmapsto e'}{\mathsf{rec}'_{\mathsf{Tree}}\, x.e_1\, x_1.x_2.e_2\, e \longmapsto \mathsf{rec}'_{\mathsf{Tree}}\, x.e_1\, x_1.x_2.e_2\, e'} \tag{10.31}$$

$$\frac{}{\mathsf{rec}'_{\mathsf{Tree}}\, x.e_1\, x_1.x_2.e_2\, (\mathsf{leaf}\, e) \longmapsto e_1[x \mapsto e]} \tag{10.32}$$

$$\frac{}{\mathsf{rec}'_{\mathsf{Tree}}\, x.e_1\, x_1.x_2.e_2\, (\mathsf{node}\, e\, e') \longmapsto e_2[x_1 \mapsto \mathsf{rec}'_{\mathsf{Tree}}\, x.e_1\, x_1.x_2.e_2\, e, x_2 \mapsto \mathsf{rec}'_{\mathsf{Tree}}\, x.e_1\, x_1.x_2.e_2\, e']} \tag{10.33}$$

Például a leveleket megszámoló függvény:

$$\lambda^{\mathsf{Tree}} y.\mathsf{rec}'_{\mathsf{Tree}}\, x.(\mathsf{suc}\,\mathsf{zero})\, x_1.x_2.x_1 + x_2\, y : \mathsf{Tree} \to \mathsf{Nat}$$

10.8. Feladat. *Írj olyan függvényt, mely függőleges hossztengelyére tükröz egy bináris fát!*

10.9. Feladat. *Add meg a természetes számokat tartalmazó listák induktív típusát (szintaxis, típusrendszer és operációs szemantika)!*

10.10. Feladat. *Add meg a természetes számok listáit összefűző függvényt!*

10.11. Feladat. *Add meg a* map *függvényt listákon, mely egy* Nat $\to$ Nat *függvényt alkalmaz pontonként a listára!*

10.12. Feladat. *Add meg azt a függvényt, mely egy bináris fából listát készít, úgy, hogy a fa leveleinél levő összes elem bekerül a listába.*

10.13. Feladat. *Add meg a leveleinél és a csomópontjainál is természetes számokat tartalmazó bináris fákat (szintaxis, típusrendszer és operációs szemantika)!*

10.14. Feladat. *Készíts olyan függvényt, mely az előző feladatban megadott bináris fából készít listát preorder, inorder és posztorder módon.*

10.15. Feladat. *NEHÉZ. Készíts olyan függvényt, mely soronként adja vissza a fa leveleit.*

A természetes számok folyama (végtelen lista, stream) egy koinduktív típus. Először megadjuk a destruktorait: kivehetjük a folyam fejét (első elemét, head) és elfelejthetjük az első elemét (a folyam farka, tail).

$$\frac{\Gamma \vdash e : \mathsf{Stream}}{\Gamma \vdash \mathsf{head}\, e : \mathsf{Nat}} \tag{10.34}$$

$$\frac{\Gamma \vdash e : \mathsf{Stream}}{\Gamma \vdash \mathsf{tail}\, e : \mathsf{Stream}} \tag{10.35}$$

Egy folyamra gondolhatunk úgy, mint egy állapotautomatára. Az aktuális állapota τ típusú. e_1 adja meg, hogy egy $x : \tau$ állapotból hogyan kapjuk meg a Nat típusú kimenetet, e_2 pedig azt, hogy az aktuális állapotból hogyan kapjuk meg a következő állapotot. A kezdőállpotot e adja meg.

$$\frac{\Gamma, x : \tau \vdash e_1 : \mathsf{Nat} \qquad \Gamma, x : \tau \vdash e_2 : \tau \qquad \Gamma \vdash e : \tau}{\Gamma \vdash \mathsf{strgen}\, x.e_1\, x.e_2\, e : \mathsf{Stream}} \tag{10.36}$$

Az operációs szemantikában egy értékünk van, egy strgen-el megadott folyam.

$$\frac{}{\mathsf{strgen}\, x.e_1\, x.e_2\, e\, \mathsf{val}} \tag{10.37}$$

A sorrendi szabályok a destruktorok paraméterét értékelik ki.

$$\frac{e \longmapsto e'}{\mathsf{head}\, e \longmapsto \mathsf{head}\, e'} \tag{10.38}$$

$$\frac{e \longmapsto e'}{\mathsf{tail}\, e \longmapsto \mathsf{tail}\, e'} \tag{10.39}$$

Az utasítás szabályok azt adják meg, hogy mi történik, ha egy destruktort alkalmazunk a generátorra. Ha a folyam fejét akarjuk megkapni, akkor az e_1-nek adjuk meg az aktuális állapotát.

$$\frac{}{\mathsf{head}\, (\mathsf{strgen}\, x.e_1\, x.e_2\, e) \longmapsto e_1[x \mapsto e]} \tag{10.40}$$

Ha a folyam farkát akarjuk megkapni, generálunk egy új folyamot, melynek aktuális állapotát e_2-vel léptetjük.

$$\frac{}{\mathsf{tail}\,(\mathsf{strgen}\,x.e_1\,x.e_2\,e) \longmapsto \mathsf{strgen}\,x.e_1\,x.e_2\,(e_2[x \mapsto e])} \qquad (10.41)$$

Például a 0,1,2, ... végtelen lista a következőképp adható meg:

$$\mathsf{strgen}\,x.x\,x.(\mathsf{suc}\,x)\,\mathsf{zero} : \mathsf{Stream}$$

Nézzük meg, mi lesz az első három eleme.

$$\begin{aligned}
&(1) && \mathsf{head}\,(\mathsf{strgen}\,x.x\,x.(\mathsf{suc}\,x)\,\mathsf{zero}) \longmapsto x[x \mapsto \mathsf{zero}] = \mathsf{zero} \\
&(2) && \mathsf{head}\,\big(\mathsf{tail}\,(\mathsf{strgen}\,x.x\,x.(\mathsf{suc}\,x)\,\mathsf{zero})\big) \\
&\longmapsto && \mathsf{head}\,(\mathsf{strgen}\,x.x\,x.(\mathsf{suc}\,x)\,(\mathsf{suc}\,x)[x \mapsto \mathsf{zero}]) = \mathsf{head}\,(\mathsf{strgen}\,x.x\,x.(\mathsf{suc}\,x)\,(\mathsf{suc}\,\mathsf{zero})) \\
&\longmapsto && x[x \mapsto \mathsf{suc}\,\mathsf{zero}] = \mathsf{suc}\,\mathsf{zero} \\
&(3) && \mathsf{head}\,\Big(\mathsf{tail}\,\big(\mathsf{tail}\,(\mathsf{strgen}\,x.x\,x.(\mathsf{suc}\,x)\,\mathsf{zero})\big)\Big) \\
&\longmapsto && \mathsf{head}\,\Big(\mathsf{tail}\,\big(\mathsf{strgen}\,x.x\,x.(\mathsf{suc}\,x)\,(\mathsf{suc}\,\mathsf{zero})\big)\Big) \\
&\longmapsto && \mathsf{head}\,\big(\mathsf{strgen}\,x.x\,x.(\mathsf{suc}\,x)\,(\mathsf{suc}\,(\mathsf{suc}\,\mathsf{zero}))\big) \\
&\longmapsto && \mathsf{suc}\,(\mathsf{suc}\,\mathsf{zero})
\end{aligned}$$

10.16. Feladat. *Add meg a* map *függvényt folyamokon, mely egy* $\mathsf{Nat} \to \mathsf{Nat}$ *függvényt alkalmaz pontonként!*

10.17. Feladat. *Meg lehet -e adni egy olyan* $\mathsf{Stream} \to \mathsf{Stream}$ *függvényt, mely tetszőleges folyamnak kiszűri az 5-nél kisebb elemeit?*

10.2.2. A fenti példák egységesített változatai

Szeretnénk az induktív és a koinduktív típusokat egységes szintaxissal megadni, úgy, hogy egyszer s mindenkorra megadjuk az összes induktív és koinduktív típust. Ehhez először megmutatjuk, hogy a fenti példákat hogyan tudjuk egységes módon megadni, az induktív esetben egy konstruktor-rekurzor, a koinduktív esetben egy destruktor-generátor párral.

Az induktív típusokat egy fold és egy rec operátorral adjuk meg. Természetes számok.

$$\frac{\Gamma \vdash e : \top + \mathsf{Nat}}{\Gamma \vdash \mathsf{fold}_{\mathsf{Nat}}\,e : \mathsf{Nat}} \qquad (10.42)$$

$$\frac{\Gamma, x : \top + \tau \vdash e_1 : \tau \qquad \Gamma \vdash e : \mathsf{Nat}}{\Gamma \vdash \mathsf{rec}_{\mathsf{Nat}}\,x.e_1\,e : \tau} \qquad (10.43)$$

$$\frac{}{\mathsf{fold}_{\mathsf{Nat}}\,e\;\mathsf{val}} \qquad (10.44)$$

$$\frac{e \longmapsto e'}{\mathsf{rec}_{\mathsf{Nat}}\,x.e_1\,e \longmapsto \mathsf{rec}_{\mathsf{Nat}}\,x.e_1\,e'} \qquad (10.45)$$

$$\frac{}{\mathsf{rec}_{\mathsf{Nat}}\,x.e_1\,(\mathsf{fold}_{\mathsf{Nat}}\,e) \longmapsto e_1[x \mapsto \mathsf{map}^{\alpha.\top+\alpha}(y.\mathsf{rec}_{\mathsf{Nat}}\,(x.e_1)\,y)\,e]} \qquad (10.46)$$

A zero konstruktor most a $\mathsf{fold}_{\mathsf{Nat}}$ operátorral és inj_1-el fejezhető ki, a suc konstruktor szintén $\mathsf{fold}_{\mathsf{Nat}}$-tal és inj_2-vel.

$$\mathsf{zero} := \mathsf{fold}_{\mathsf{Nat}}\,(\mathsf{inj}_1\,\mathsf{tt}) : \mathsf{Nat}$$

$$\frac{n : \mathsf{Nat}}{\mathsf{suc}\, n := \mathsf{fold}_{\mathsf{Nat}}\, (\mathsf{inj}_2\, n) : \mathsf{Nat}}$$

A rekurzorban (10.43) az e_1 paraméter egyszerre adja meg, hogy mit kell nulla és rákövetkező esetben tenni. Nulla esetén nem kap semmi információt ($\top$), rákövetkező esetén megkapja a rekurzív hívás τ típusú eredményét. A $\mathsf{fold}_{\mathsf{Nat}}$ konstruktor alkalmazása értéket ad meg (10.44), és a sorrendi szabály (10.45) a rekurzor természetes szám paraméterét értékeli ki. Ha a rekurzort a konstruktorra alkalmazzuk (10.46), akkor e_1-re írjuk át az eredményt, ami függ egy $x : \top + \tau$ változótól. Ennek értékét úgy adjuk meg, hogy a generikus map segítségével a τ részt helyettesítjük a rekurzív hívással. Ha ezt átírjuk a generikus programozás 10.12., 10.9 és 10.8 szabályait alkalmazva, a következőt kapjuk.

$$\frac{}{\mathsf{rec}_{\mathsf{Nat}}\, x.e_1\, (\mathsf{fold}_{\mathsf{Nat}}\, e) \longmapsto e_1[x \mapsto \mathsf{case}\, e\, x_1.(\mathsf{inj}_1\, x_1)\, x_2.(\mathsf{inj}_2\, (\mathsf{rec}_{\mathsf{Nat}}\, (x.e_1)\, x_2))]}$$

Tehát case-szel megnézzük, hogy a konstruktor e paramétere nullát vagy rákövetkezőt jelent -e. Az előbbi esetben x-et $\mathsf{inj}_1\, \mathsf{tt}$-vel helyettesítjük ($\top$ egyetlen eleme tt), utóbbi esetben a rekurzív hívásra inj_2-t alkalmazunk, és ezzel helyettesítjük x-et.

A duplázás függvény most így adható meg az előbbi zero és suc rövidítéseket felhasználva:

$$\mathsf{dup} := \lambda^{\mathsf{Nat}} x.\mathsf{rec}_{\mathsf{Nat}}\, y.(\mathsf{case}\, y\, x_1.\mathsf{zero}\, x_2.\mathsf{suc}\, (\mathsf{suc}\, x_2))\, x : \mathsf{Nat} \to \mathsf{Nat}$$

Esetszétválasztást (case) használtunk aszerint, hogy zero (inj_1) vagy suc (inj_2) volt az y természetes szám, amit duplázni akarunk. A zero esetben zero-t adunk vissza, a suc esetben x_2 jelöli n-nek a dupláját, így ($\mathsf{suc}\, n$) duplája $\mathsf{suc}\, (\mathsf{suc}\, x_2)$ lesz.

Bináris fák (megjegyezzük, hogy $\tau_1 + \tau_2 \times \tau_3$ zárójelezése $\tau_1 + (\tau_2 \times \tau_3)$).

$$\frac{\Gamma \vdash e : \mathsf{Nat} + \mathsf{Tree} \times \mathsf{Tree}}{\Gamma \vdash \mathsf{fold}_{\mathsf{Tree}}\, e : \mathsf{Tree}} \tag{10.47}$$

$$\frac{\Gamma, x : \mathsf{Nat} + \tau \times \tau \vdash e_1 : \tau \qquad \Gamma \vdash e : \mathsf{Tree}}{\Gamma \vdash \mathsf{rec}_{\mathsf{Tree}}\, x.e_1\, e : \tau} \tag{10.48}$$

$$\frac{}{\mathsf{fold}_{\mathsf{Tree}}\, e\, \mathsf{val}} \tag{10.49}$$

$$\frac{e \longmapsto e'}{\mathsf{rec}_{\mathsf{Tree}}\, x.e_1\, e \longmapsto \mathsf{rec}_{\mathsf{Tree}}\, x.e_1\, e'} \tag{10.50}$$

$$\frac{}{\mathsf{rec}_{\mathsf{Tree}}\, x.e_1\, (\mathsf{fold}_{\mathsf{Tree}}\, e) \longmapsto e_1[x \mapsto \mathsf{map}^{\alpha.\mathsf{Nat}+\alpha\times\alpha}(y.\mathsf{rec}_{\mathsf{Tree}}\, (x.e_1)\, y)\, e]} \tag{10.51}$$

Természetes számok folyamai. Az destuktorokat az unfold operátorba vontuk össze. A generátort gen-el jelöljük. Ha meg van adva egy folyam, abból meg tudunk adni egy természetes számot (korábban head) és egy újabb folyamot (korábban tail).

$$\frac{\Gamma \vdash e : \mathsf{Stream}}{\Gamma \vdash \mathsf{unfold}_{\mathsf{Stream}}\, e : \mathsf{Nat} \times \mathsf{Stream}} \tag{10.52}$$

A folyam generálásához kell egy e_1, ami egy adott $x : \tau$ állapotra megadja a kimenetet (egy természetes szám) és az új állapotot (τ). Ezenkívül szükségünk van az aktuális állapotra, ezt e adja meg.

$$\frac{\Gamma, x : \tau \vdash e_1 : \mathsf{Nat} \times \tau \qquad \Gamma \vdash e : \tau}{\Gamma \vdash \mathsf{gen}_{\mathsf{Stream}}\, x.e_1\, e : \mathsf{Stream}} \tag{10.53}$$

Egy generált folyam érték.

$$\frac{}{\mathsf{gen}_{\mathsf{Stream}}\, x.e_1\, e\; \mathsf{val}} \tag{10.54}$$

A sorrendi szabály azt mondja, hogy a destruktor paraméterét kiértékelhetjük.

$$\frac{e \longmapsto e'}{\mathsf{unfold}_{\mathsf{Stream}}\, e \longmapsto \mathsf{unfold}_{\mathsf{Stream}}\, e'} \tag{10.55}$$

Ha a destruktort a generátorra alkalmazzuk, akkor e_1-nek megadjuk az aktuális állapotot (e-t), és ezzel megkapunk egy kimenet—új állapot párt. Nekünk viszont egy kimenet—új folyam párra van szükségünk, ezért a generikus map segítségével az új állapotra meghívjuk a $\mathsf{gen}_{\mathsf{Stream}}\, x.e_1$ generátort.

$$\frac{}{\mathsf{unfold}_{\mathsf{Stream}}\, (\mathsf{gen}_{\mathsf{Stream}}\, x.e_1\, e) \longmapsto \mathsf{map}^{\alpha.\mathsf{Nat}\times\alpha}(y.\mathsf{gen}_{\mathsf{Stream}}\, x.e_1\, y)\, (e_1[x \mapsto e])} \tag{10.56}$$

10.2.3. Általános induktív és koinduktív típusok

Ennyi példa után megadjuk az általános esetet. A véges típusokat és függvényeket tartalmazó nyelvet egészítünk ki induktív és koinduktív típusokkal.

A típusok lehetnek típusváltozók (α, α', β stb.) vagy induktív vagy koinduktív típusok. ind és coind aritása $(\mathsf{Ty}.\mathsf{Ty})\mathsf{Ty}$. Tehát egy konkrét induktív vagy koinduktív típus megadásához meg kell adni egy $\alpha.\tau$ típusoperátort. Természetes számok esetén ez pl. $\alpha.\top + \alpha$ lesz, bináris fák esetén $\beta.\mathsf{ind}^{\alpha.\top+\alpha} + \beta \times \beta$, folyamok esetén $\beta.\mathsf{ind}^{\alpha.\top+\alpha} \times \beta$.

$$\tau, \tau_1, ... \in \mathsf{Ty} ::= ... \mid \alpha \mid \mathsf{ind}^{\alpha.\tau} \mid \mathsf{coind}^{\alpha.\tau} \tag{10.57}$$

A szintaxist a konstruktorral-rekurzorral és a destruktorral-generátorral egészítjük ki.

$$\begin{aligned} e, e', ... \in \mathsf{Exp} ::= ... &\mid \mathsf{fold}^{\alpha.\tau}\, e && \text{konstruktor} \\ &\mid \mathsf{rec}^{\alpha.\tau}\, e && \text{rekurzor} \\ &\mid \mathsf{unfold}^{\alpha.\tau}\, e && \text{destruktor} \\ &\mid \mathsf{gen}^{\alpha.\tau}\, e && \text{generátor} \end{aligned} \tag{10.58}$$

Típusrendszer. Induktív típus egy elemét a fold konstruktorral adjuk meg. A rekurzív előfordulásokat a τ típusban az α-k jelölik, ezeket magával az induktív típussal helyettesítjük.

$$\frac{\alpha.\tau\; \mathsf{spos} \qquad \Gamma \vdash e : \tau[\alpha \mapsto \mathsf{ind}^{\alpha.\tau}]}{\Gamma \vdash \mathsf{fold}^{\alpha.\tau}\, e : \mathsf{ind}^{\alpha.\tau}} \tag{10.59}$$

A rekurzornak meg kell adni egy e induktív típusbeli elemet, és hogy hogyan működjön a különböző konstrukciós formák esetén, melyeket τ ad meg. Most τ-ban az α típusváltozót τ'-vel helyettesítjük, ebbe a típusba eliminálunk, és emiatt a rekurzív hívások eredményei is ilyen típusúak.

$$\frac{\alpha.\tau\; \mathsf{spos} \qquad \Gamma, x : \tau[\alpha \mapsto \tau'] \vdash e_1 : \tau' \qquad \Gamma \vdash e : \mathsf{ind}^{\alpha.\tau}}{\Gamma \vdash \mathsf{rec}^{\alpha.\tau}\, x.e_1\, e : \tau'} \tag{10.60}$$

Egy koinduktív típusú e kifejezésből a destruktor segítségével megkapjuk a τ típusú értékeket, ahol az α típusváltozó a koinduktív típussal van helyettesítve.

$$\frac{\alpha.\tau\,\mathsf{spos} \qquad \Gamma \vdash e : \mathsf{coind}^{\alpha.\tau}}{\Gamma \vdash \mathsf{unfold}^{\alpha.\tau}\, e : \tau[\alpha \mapsto \mathsf{coind}^{\alpha.\tau}]} \tag{10.61}$$

Egy koinduktív típusú elemet úgy generálunk, hogy megadjuk az aktuális állapotát ($e : \tau'$), és hogy hogy mit adnak a destruktorok attól függően, hogy mi az aktuális állapot.

$$\frac{\alpha.\tau\,\mathsf{spos} \qquad \Gamma, x : \tau' \vdash e_1 : \tau[\alpha \mapsto \tau'] \qquad \Gamma \vdash e : \tau'}{\Gamma \vdash \mathsf{gen}^{\alpha.\tau}\, x.e_1\, e : \mathsf{coind}^{\alpha.\tau}} \tag{10.62}$$

Figyeljük meg az induktív és koinduktív típusokra vonatkozó szabályok szimmetriáját.

Lusta operációs szemantika. Az egyetlen induktív érték a konstuktor.

$$\frac{}{\mathsf{fold}^{\alpha.\tau}\, e\,\mathsf{val}} \tag{10.63}$$

A rekurzor induktív típusú paraméterét kiértékeljük.

$$\frac{e \longmapsto e'}{\mathsf{rec}^{\alpha.\tau}\, x.e_1\, e \longmapsto \mathsf{rec}^{\alpha.\tau}\, x.e_1\, e'} \tag{10.64}$$

Ha a rekurzort egy konstruktorra alkalmazzuk, akkor a rekurzor e_1 paraméterét használjuk, ami megmondja, hogy mely konstrukciós forma esetén mit kell csinálni, és ez függ x-től, amibe e-t, a konstrukciós formát magát szeretnénk helyettesíteni. e-ben viszont a rekurzív előfordulásokat (ahol τ-ban α van) helyettesíteni kell a rekurzív hívással.

$$\frac{}{\mathsf{rec}^{\alpha.\tau}\, x.e_1\, (\mathsf{fold}^{\alpha.\tau}\, e) \longmapsto e_1[x \mapsto \mathsf{map}^{\alpha.\tau}\, y.(\mathsf{rec}^{\alpha.\tau}\, x.e_1\, y)\, e]} \tag{10.65}$$

Az egyetlen koinduktív érték a generátor.

$$\frac{}{\mathsf{gen}^{\alpha.\tau}\, x.e_1\, e\,\mathsf{val}} \tag{10.66}$$

A destruktor paraméterét kiértékeljük.

$$\frac{e \longmapsto e'}{\mathsf{unfold}^{\alpha.\tau}\, e \longmapsto \mathsf{unfold}^{\alpha.\tau}\, e} \tag{10.67}$$

Ha a destruktort a generátorra alkalmazzuk, akkor e_1-et adjuk vissza, mely megadja az eredményt minden destrukciós formára. Viszont ez függ az x-től, melyet az aktuális állapot, $e : \tau'$ ad meg. A rekurzív előfordulások viszont koinduktív típusúak kell, hogy legyen, nem τ' típusúak, ezért a generikus map segítségével az α helyeken újabb koinduktív típusokat generálunk az új τ' állapotokból.

$$\frac{}{\mathsf{unfold}^{\alpha.\tau}\, (\mathsf{gen}^{\alpha.\tau}\, x.e_1\, e) \longmapsto \mathsf{map}^{\alpha.\tau}\, (y.\mathsf{gen}^{\alpha.\tau}\, x.e_1\, y)(e_1[x \mapsto e])} \tag{10.68}$$

10.18. Feladat. *Bővítsük ki a generikus* map *operátort, hogy induktív és koinduktív típusokon is működjön!*

További információ:

– Az $\alpha.\tau$ spos ítéletet a következő levezetési szabályokkal adjuk meg (melyek egyszerre vannak megadva a τ type levezetési szabályaival).

$$\frac{}{\alpha.\top\ \mathsf{spos}} \tag{10.69}$$

$$\frac{\alpha.\tau_1\ \mathsf{spos} \qquad \alpha.\tau_2\ \mathsf{spos}}{\alpha.\tau_1 \times \tau_2\ \mathsf{spos}} \tag{10.70}$$

$$\frac{}{\alpha.\bot\ \mathsf{spos}} \tag{10.71}$$

$$\frac{\alpha.\tau_1\ \mathsf{spos} \qquad \alpha.\tau_2\ \mathsf{spos}}{\alpha.\tau_1 + \tau_2\ \mathsf{spos}} \tag{10.72}$$

$$\frac{\tau_1\ \mathsf{type} \qquad \alpha.\tau_2\ \mathsf{spos}}{\alpha.\tau_1 \rightarrow \tau_2\ \mathsf{spos}} \tag{10.73}$$

$$\frac{\mathsf{ind}^{\beta.\tau}\ \mathsf{type}}{\alpha.\mathsf{ind}^{\beta.\tau}\ \mathsf{spos}} \tag{10.74}$$

$$\frac{\mathsf{coind}^{\beta.\tau}\ \mathsf{type}}{\alpha.\mathsf{coind}^{\beta.\tau}\ \mathsf{spos}} \tag{10.75}$$

$$\frac{}{\top\ \mathsf{type}} \tag{10.76}$$

$$\frac{\tau_1\ \mathsf{type} \qquad \tau_2\ \mathsf{type}}{\tau_1 \times \tau_2\ \mathsf{type}} \tag{10.77}$$

$$\frac{}{\bot\ \mathsf{type}} \tag{10.78}$$

$$\frac{\tau_1\ \mathsf{type} \qquad \tau_2\ \mathsf{type}}{\tau_1 + \tau_2\ \mathsf{type}} \tag{10.79}$$

$$\frac{\tau_1\ \mathsf{type} \qquad \tau_2\ \mathsf{type}}{\tau_1 \rightarrow \tau_2\ \mathsf{type}} \tag{10.80}$$

$$\frac{\alpha.\tau\ \mathsf{spos}}{\mathsf{ind}^{\alpha.\tau}\ \mathsf{type}} \tag{10.81}$$

$$\frac{\alpha.\tau\ \mathsf{spos}}{\mathsf{coind}^{\alpha.\tau}\ \mathsf{type}} \tag{10.82}$$

– Ha $\alpha.\tau$-t nem szorítjuk meg szigorúan pozitív típuskifejezésekre, hanem negatív típuskifejezéseket is megengedünk, akkor az üres környezetben meg tudunk adni $\bot$ típusú elemet:

$$\begin{aligned}
\cdot \vdash\ & \mathsf{let}\ \lambda^{\bot} x.x\ \mathsf{in}\ f.\\
& \quad \mathsf{let}\ \lambda^{\mathsf{ind}^{\alpha.\alpha\rightarrow\bot}} y.\mathsf{rec}^{\alpha.\alpha\rightarrow\bot}\ x.x\,y\ \mathsf{in}\ app.\\
& \qquad \mathsf{let}\ \mathsf{fold}^{\alpha.\alpha\rightarrow\bot}\ (\lambda^{\mathsf{ind}^{\alpha.\alpha\rightarrow\bot}} x.f\ (app\,x\,x))\ \mathsf{in}\ h.\\
& \qquad\quad app\,h\,h \qquad\qquad\qquad\qquad : \bot
\end{aligned}$$

10.2.4. További példák

A környezetfüggetlen nyelvtannal megadott formális nyelvek induktív típusok.

Pl. vegyük az alábbi nyelvtant, ahol S és A a nemterminálisok (nyelvtani szimbólumok), a, b a terminálisok.

$$\begin{aligned} A &\to bA \mid a \\ S &\to aSA \mid A \end{aligned}$$

A következő rövidítésekkel adható meg induktív típusokkal és konstruktorokkal.

$$\begin{aligned}
A \quad &:= \mathsf{ind}^{\alpha.\alpha+\top} \\
S \quad &:= \mathsf{ind}^{\alpha.\alpha\times A+A} \\
b- \quad &:= \lambda^A x.\mathsf{fold}^{\alpha.\alpha+\top}\,(\mathsf{inj}_1\,x) \\
a \quad &:= \mathsf{fold}^{\alpha.\alpha+\top}\,(\mathsf{inj}_2\,\mathsf{tt}) \\
a-- \quad &:= \lambda^S x.\lambda^A y.\mathsf{fold}^{\alpha.\alpha\times A+A}\,(\mathsf{inj}_1\,\langle x, y\rangle) \\
- \quad &:= \lambda^A x.\mathsf{fold}^{\alpha.\alpha\times A+A}\,(\mathsf{inj}_2\,x)
\end{aligned}$$

Néhány példa induktív típusokra:

$\mathsf{ind}^{\alpha.\top+\alpha}$	természetes számok
$\mathsf{ind}^{\alpha.\top+\tau\times\alpha}$	τ típusú elemeket tartalmazó listák
$\mathsf{ind}^{\alpha.\top+\alpha\times\alpha}$	bináris fák, a leveleknél nincs információ
$\mathsf{ind}^{\alpha.\top+(\mathsf{Nat}\to\alpha)}$	minden lépésben végtelen-felé ágazó fák
$\mathsf{ind}^{\alpha.\top+(\alpha\times\alpha)+(\alpha\times\alpha\times\alpha)}$	kétfelé és háromfelé is ágazó fák
$\mathsf{ind}^{\alpha.\mathsf{Nat}+\mathsf{int}+\mathsf{str}+\alpha\times\alpha+\alpha\times\alpha+\alpha\times\alpha+\alpha+\alpha\times\mathsf{Nat}\times\alpha}$	számok és szövegek kifejezései, változók helyett természetes számok (De Bruijn indexek)

10.19. Feladat. *Adjuk meg a Turing-gépet, mint koinduktív típust!*

További információ:

– A kölcsönösen megadott környezetfüggetlen nyelvtanok is leírhatók induktív típusokkal, de ezekhez kölcsönös induktív típusok vagy indexelt induktív típusok kellenek, ezeket nem tárgyaljuk. Ilyen nyelvtan pl.:

$$\begin{aligned} A &\to bA \mid a \mid cS \\ S &\to aSA \mid A \end{aligned}$$

11. Polimorfizmus

A típusokat arra hasznátuk, hogy kiszűrjük az értelmetlen programokat. Ennek viszont van egy ára: kód-ismétlődés. Például az identitás függvénynek annyi féle definíciója van, ahány típus: $\lambda^{\mathsf{int}}x.x : \mathsf{int} \to \mathsf{int}$, $\lambda^{\mathsf{str}}x.x : \mathsf{str} \to \mathsf{str}$, $\lambda^{\mathsf{str}\to\mathsf{int}}x.x : (\mathsf{str} \to \mathsf{int}) \to (\mathsf{str} \to \mathsf{int})$ stb. Hasonlóan, a függvénykompozíció operátort is külön-külön kell megadnunk minden τ_1, τ_2, τ_3 típushármasra:

$$\lambda^{\tau_2\to\tau_3}g.\lambda^{\tau_1\to\tau_2}f.\lambda^{\tau_1}x.g\,(f\,x) : (\tau_2 \to \tau_3) \to (\tau_1 \to \tau_2) \to (\tau_1 \to \tau_3)$$

A megoldás az, hogy típusváltozókat használunk az olyan típusok helyett, amik sokfélék (polimorfak) lehetnek (ahhoz hasonlóan, ahogy az előző fejezetben a típusoperátorokat egy típusváltozóval adtuk meg). Például $\lambda^\alpha x.x : \alpha \to \alpha$, ahol α-t tetszőleges típussal helyettesíthetjük.

11.1. Szintaxis

$$\begin{aligned}
\tau, \tau_1, \tau', ... \in \mathsf{Ty} &::= \alpha \,|\, \tau_1 \to \tau_2 \,|\, \forall\alpha.\tau \\
e, e_1, e', ... \in \mathsf{Exp} &::= x \,|\, \lambda^\tau x.e \,|\, e\,e' \,|\, \Lambda\alpha.e \,|\, e[\tau]
\end{aligned}$$

$\Lambda\alpha.e$ egy polimorf kifejezés, mely bármely bemeneti típusra egységes módon működik. Típusa $\forall\alpha.\tau$ lesz, ahol τ-ban α kötve van ($\forall$ aritása $(\mathsf{Ty}.\mathsf{Ty})\mathsf{Ty}$). Egy ilyen típusú polimorf kifejezésre alkalmazni tudunk egy típust a típusalkalmazás $e[\tau]$ operátorral, aritása $(\mathsf{Exp}, \mathsf{Ty})\mathsf{Exp}$.

11.2. Típusrendszer

A jólformált típus ítélet $\Delta \vdash \tau$ alakú. Ebben Δ egy típusváltozók környezete, azaz típusváltozók listája.

$$\Delta, \Delta', ... \in \mathsf{TCon} ::= \cdot \,|\, \Delta, \alpha \tag{11.1}$$

Típusváltozók környezetének kényelmes kezeléséhez kellenek az alábbi definíciók.

$$\begin{aligned}
\mathsf{dom}(\cdot) &:= \{\} \\
\mathsf{dom}(\Delta, \alpha) &:= \{\alpha\} \cup \mathsf{dom}(\Delta)
\end{aligned} \tag{11.2}$$

$$\frac{}{\cdot\,\mathsf{wf}} \tag{11.3}$$

$$\frac{\Delta\,\mathsf{wf} \qquad \alpha \notin \mathsf{dom}(\Delta)}{\Delta, \alpha\,\mathsf{wf}} \tag{11.4}$$

$$\frac{\Delta\,\mathsf{wf} \qquad \alpha \notin \mathsf{dom}(\Delta)}{\alpha \in \Delta, \alpha} \tag{11.5}$$

$$\frac{\alpha \in \Delta \qquad \alpha \notin \mathsf{dom}(\Delta)}{\alpha \in \Delta, \alpha'} \tag{11.6}$$

A jólformált típus ítélet levezetési szabályai.

$$\frac{\alpha \in \Delta}{\Delta \vdash \alpha} \tag{11.7}$$

$$\frac{\Delta \vdash \tau_1 \qquad \Delta \vdash \tau_2}{\Delta \vdash \tau_1 \to \tau_2} \tag{11.8}$$

$$\frac{\Delta, \alpha \vdash \tau}{\Delta \vdash \forall\alpha.\tau} \tag{11.9}$$

Kifejezések típusozási ítélete mosantól hivatkozik egy típuskörnyezetre is, tehát $\Delta\,\Gamma \vdash e : \tau$ alakú.

$$\frac{(x : \tau) \in \Gamma}{\Delta\,\Gamma \vdash x : \tau} \tag{11.10}$$

$$\frac{\Delta\,\Gamma, x : \tau_1 \vdash e : \tau_2}{\Delta\,\Gamma \vdash \lambda^{\tau_1} x.e : \tau_1 \to \tau_2} \tag{11.11}$$

$$\frac{\Delta\,\Gamma \vdash e : \tau_1 \to \tau_2 \qquad \Delta\,\Gamma \vdash e : \tau_1}{\Delta\,\Gamma \vdash e\, e_1 : \tau_2} \tag{11.12}$$

$$\frac{\Delta, \alpha\,\Gamma \vdash e : \tau}{\Delta\,\Gamma \vdash \Lambda\alpha.e : \forall\alpha.\tau} \tag{11.13}$$

$$\frac{\Delta\,\Gamma \vdash e : \forall\alpha.\tau \qquad \Delta \vdash \tau_1}{\Delta\,\Gamma \vdash e[\tau_1] : \tau[\alpha \mapsto \tau_1]} \tag{11.14}$$

A polimorf identitás-függvény most már megadható a következőképp:

$$\Lambda\alpha.\lambda^{\alpha} x.x : \forall\alpha.\alpha \to \alpha$$

A gyakorlatban pedig használhatjuk többféle típusra (ehhez a példához a fenti nyelvet kibővítjük int, str típusokkal és let-tel):

$$\mathsf{let}\ \Lambda\alpha.\lambda^{\alpha} x.x\ \mathsf{in}\ I.|I[\mathsf{str}]\,\text{"}\mathsf{abc}\text{"}| + (I[\mathsf{int}]\,3)$$

A polimorf függvénykompozíció:

$$\Lambda\alpha_1.\Lambda\alpha_2.\Lambda\alpha_3.\lambda^{\alpha_2\to\alpha_3} g.\lambda^{\alpha_1\to\alpha_2} f.\lambda^{\alpha_1} x.g\,(f\,x) : \forall\alpha_1.\forall\alpha_2.\forall\alpha_3.(\alpha_2 \to \alpha_3) \to (\alpha_1 \to \alpha_2) \to (\alpha_1 \to \alpha_3)$$

11.3. Operációs szemantika

A szögletes zárójelbe tett szabályok érték szerinti paraméterátadást esetén veendők figyelembe. Ha nincsenek, akkor név szerinti paraméterátadás történik.

$$\frac{}{\lambda^{\tau} x.e\ \mathsf{val}} \tag{11.15}$$

$$\frac{}{\Lambda\alpha.e\ \mathsf{val}} \tag{11.16}$$

$$\frac{[e_1\ \mathsf{val}]}{(\lambda^{\tau}\, x.e)\, e_1 \longmapsto e[x \mapsto e_1]} \tag{11.17}$$

$$\frac{e \longmapsto e'}{e\, e_1 \longmapsto e'\, e_1} \tag{11.18}$$

$$\left[\frac{e\ \mathsf{val} \qquad e_1 \longmapsto e_1'}{e\, e_1 \longmapsto e\, e_1'}\right] \tag{11.19}$$

$$\frac{}{(\Lambda\,\alpha.e)[\tau_1] \longmapsto e[\alpha \mapsto \tau_1]} \tag{11.20}$$

$$\frac{e \longmapsto e'}{e[\tau_1] \longmapsto e'[\tau_1]} \tag{11.21}$$

A haladás és típusmegőrzés tétele bebizonyítható.

11.4. Absztrakció

Absztrakt típusok (egzisztenciális típus, interface) is megadhatók polimorf típusokkal. Például ha τ_s egy (Nat-okat tartalmazó) verem megvalósítása, akkor a következő szolgáltatásokat várjuk el a veremtől:

empty	$: \tau_s$	létrehoz egy üres vermet
push	$: \mathsf{Nat} \to \tau_s \to \tau_s$	verembe rakás
pop	$: \tau_s \to (\mathsf{Opt}\,\mathsf{Nat} \times \tau_s)$	verem legfelső elemének kiszedése
$\mathsf{isEmpty}$	$: \tau_s \to \mathsf{Bool}$	megadja, hogy a verem üres -e

Egy olyan kifejezés típusa, mely egy (absztrakt) vermet használ, de nem függ a verem implementációjától, sőt, bármely verem-implementációra egységesen működik, a követekezőképp adható meg.

$$\forall\alpha.\Big(\alpha \times (\mathsf{Nat} \to \alpha \to \alpha) \times \big(\alpha \to (\mathsf{Opt}\,\mathsf{Nat} \times \alpha)\big) \times (\alpha \to \mathsf{Bool})\Big) \to \alpha$$

Ezt úgy is szokták hívni, hogy ez egy interface-szel van paraméterezve. A $- \times \times -$ jobbra zárójeleződik. Az α jelöli a verem implementációjának a típusát, a kifejezés következő paramétere maga az implementáció, majd visszaad egy ilyen vermet. A következő kifejezés annyit csinál, hogy létrehoz egy üres vermet:

$$\Lambda\alpha.\lambda^{\alpha\times(\mathsf{Nat}\to\alpha\to\alpha)\times\big(\alpha\to(\mathsf{Opt}\,\mathsf{Nat}\times\alpha)\big)\times(\alpha\to\mathsf{Bool})}stack.\mathsf{proj}_1\,stack$$

A következő kifejezés egy üres verembe beteszi az 1,2,3 számokat, majd kiveszi a legfelső elemet, és visszaadja az így kapott vermet:

$$\begin{aligned}
&\Lambda\alpha.\lambda^{\alpha\times(\mathsf{Nat}\to\alpha\to\alpha)\times\big(\alpha\to(\mathsf{Opt}\,\mathsf{Nat}\times\alpha)\big)\times(\alpha\to\mathsf{Bool})}stack\\
&\qquad.\mathsf{let}\ \mathsf{proj}_1(\mathsf{proj}_2\,stack)\ \mathsf{in}\,push\\
&\qquad.\mathsf{let}\ \mathsf{proj}_1\,\big(\mathsf{proj}_2\,(\mathsf{proj}_2\,stack)\big)\ \mathsf{in}\,pop\\
&\qquad.\mathsf{proj}_2\left(pop\left(push\,3\,\Big(push\,2\,\big(push\,1\,(\mathsf{proj}_1\,stack)\big)\Big)\right)\right)
\end{aligned}$$

Hány féle elemei lehetnek az alábbi típusoknak?

$$\begin{aligned}
&\forall\alpha.\alpha \to \alpha\\
&\forall\alpha.\alpha \to (\alpha \to \alpha)\\
&\forall\alpha.(\alpha \to \alpha) \to \alpha\\
&\forall\alpha.(\alpha \to \alpha) \to (\alpha \to \alpha)
\end{aligned}$$

Anélkül, hogy ránéznénk a kifejezésre magára, a típusából megtudjuk, hogy bizonyos tulajdonságoknak megfelel: ezt hívjak parametricitásnak, absztrakciónak vagy ingyen tételnek (parametricity, abstraction theorem, free theorem). Megnézünk egy típust, és ingyen kapunk egy tételt, ami az összes, ilyen típusú programra igaz lesz.

Például egy $\forall\alpha.\alpha \to \alpha$ típusú kifejezés először kap egy tetszőleges τ típust (mely α helyére lesz behelyettesítve), majd egy ilyen τ típusú elemet, és vissza kell adnia egy τ típusú elemet. Milyen elemet tudunk visszaadni? A τ típusról nem tudunk semmit: például nem tudjuk, hogy lehet -e összeadni τ típusú elemeket, vagy lehet -e esetet szétválasztani rájuk, sőt, még azt sem tudjuk, hogy egyáltalán van -e τ-nak eleme (hiszen lehet például $\perp$ is). Emiatt csak egyféle dolgot csinálhatunk: visszaadjuk a bemeneten kapott elemet. Tehát $\forall\alpha.\alpha \to \alpha$ típusú csak az identitás függvény lehet, tehát az alábbi kifejezés:

$$\cdot \vdash \Lambda\alpha.\lambda^\alpha x.x : \forall\alpha.\alpha \to \alpha$$

Persze írhatunk más kifejezést is, pl. a $\Lambda\alpha.\lambda^\alpha x.(\lambda^\alpha y.y)\,x$ kifejezés is ugyanilyen típusú lesz, de ez a kifejezés ugyanúgy viselkedik, mint az előző: ugyanarra a bemenetre ugyanazt a kimenetet adja. Ezért azt mondjuk, hogy $\forall\alpha.\alpha \to \alpha$-nak viselkedés szerint csak egyféle eleme van.

A parametricitás azt mondja, hogy a típusozott kifejezések tetszőleges relációt megtartanak. Például egy $e : \forall\alpha.\alpha \to \alpha$ kifejezésre a következőt állítja.

11.1. Tétel (Parametricitás $e : \forall\alpha.\alpha \to \alpha$-ra)**.** *Tfh. van egy R relációnk τ_1 és τ_2 típusú kifejezések között, mely nem különböztet meg egymásba átírható kifejezéseket, tehát ha $R\,e_1\,e_2$ és $e_1 \longmapsto^* e_1'$ és $e_2 \longmapsto^* e_2'$, akkor $R\,e_1'\,e_2'$. Ekkor, ha $R\,e_1\,e_2$, akkor $R\,(e[\tau_1]\,e_1)\,(e[\tau_2]\,e_2)$.*

Ezt a tételt felhasználva beláthatjuk, hogy egy $\forall\alpha.\alpha \to \alpha$ típusú e az identitás kell, hogy legyen. Például, ha $\tau_1 = \mathsf{int}$, akkor belátjuk, hogy $e[\mathsf{int}]\,0 \longmapsto^* 0$. Ehhez úgy adjuk meg az R relációt, hogy $R\,e_1\,e_1 = (e_1 \longmapsto^* 0)$ (a második paraméterét nem veszi figyelembe). Ekkor igaz, hogy $R\,0\,e_2$, mert ez azt jelenti, hogy $0 \longmapsto^* 0$, és emiatt igaz, hogy $R\,(e[\mathsf{int}]\,0)\,(e[\tau_2]\,e_2)$, tehát $(e[\mathsf{int}]\,0) \longmapsto^* 0$.

Hasonló parametricitás (relációk megtartása) tételek bizonyíthatók mindenféle típusra.

További információ:

- Ezt a típusrendszert System F-nek hívják, Girard és Reynolds adta meg először, egymástól függetlenül.
- System F-ben elkódolhatók az induktív és koinduktív típusok, nem kell külön bevezetnünk őket.

12. Altípus

Harper könyv: X. rész.

Egy τ típust tekinthetünk egy τ' típus altípusának, ha minden olyan konstruktorra, mely τ típusú elemet készít, alkalmazni tudjuk τ' összes eliminátorát.

Példa: egy természetes számot tudunk egész számnak tekinteni. Ezt úgy jelöljük, hogy $\mathsf{Nat} < \mathsf{int}$. Ezt a $\mathbb{N} \subset \mathbb{Z}$ matematikai beágyazás támasztja alá. És bevezethetjük az alábbi szabályt:

$$\frac{\Gamma \vdash e : \mathsf{Nat} \qquad \mathsf{Nat} < \mathsf{int}}{\Gamma \vdash e : \mathsf{int}}$$

Ezáltal a jól típusozott kifejezések körét bővítjük. Mostantól ha egy int-et várunk, írhatunk Nat-ot is.

Általánosabban, bevezetünk egy új ítéletet, ami $\tau < \tau'$ alakú, és a típusrendszert kiegészítjük az alábbi levezetési szabállyal.

$$\frac{\Gamma \vdash e : \tau \qquad \tau < \tau'}{\Gamma \vdash e : \tau'} \tag{12.1}$$

Általános szorzat típusokra: az eliminációs forma az i_k. projekció, ezt végre tudjuk hajtani, ha több komponens van a szorzatban, tehát ha az altípus indexelő halmaza nagyobb, mint a bővebb típus indexelő halmaza.

$$\frac{\{j_1, ..., j_m\} \subset \{i_1, ..., i_n\}}{\Pi(i_1 \hookrightarrow \tau_{i_1}, ..., i_n \hookrightarrow \tau_{i_n}) < \Pi(j_1 \hookrightarrow \tau_{j_1}, ..., j_m \hookrightarrow \tau_{j_m})} \tag{12.2}$$

Általános összeg típusokra az eliminátor a `case` esetszétválasztás, ezt alkalmazni tudjuk mindig, ha a konstruktorok indexei a bővebb típus konstruktorainak egy részhalmaza.

$$\frac{\{i_1, ..., i_n\} \subset \{j_1, ..., j_m\}}{\Sigma(i_1 \hookrightarrow \tau_{i_1}, ..., i_n \hookrightarrow \tau_{i_n}) < \Sigma(j_1 \hookrightarrow \tau_{j_1}, ..., j_m \hookrightarrow \tau_{j_m})} \tag{12.3}$$

Minden típus altípusa $\top$-nak (hiszen $\top$-nak nincs egy eliminátora sem, tehát minden eliminátora alkalmazható minden típusra).

$$\frac{}{\tau < \top} \tag{12.4}$$

A $\bot$ típus pedig minden más típus altípusának, mert nincs egy konstruktora sem, ezért az összes konstruktorára tudjuk alkalmazni bármely más típus eliminátorát. Az alábbi szabályt 12.1-gyel kombinálva megkapjuk a $\bot$ típus 7.25. eliminációs szabályát.

$$\frac{}{\bot < \tau} \tag{12.5}$$

Minden függvény, mely egy τ_2-beli kifejezést ad vissza, visszaad egy τ_2'-beli kifejejezést is, feltéve, hogy $\tau_2 < \tau_2'$.

$$\frac{\tau_2 < \tau_2'}{(\tau_1 \to \tau_2) < (\tau_1 \to \tau_2')} \tag{12.6}$$

Egy $\tau_1 \to \tau_2$-beli függvény bármely τ_1 bemenetre működik, így egy τ_1'-beli bemenetre is működik, ha $\tau_1' < \tau_1$.

$$\frac{\tau_1' < \tau_1}{(\tau_1 \to \tau_2) < (\tau_1' \to \tau_2)} \tag{12.7}$$

A kettőt kombinálva a következő szabályt kapjuk:

$$\frac{\tau_1' < \tau_1 \qquad \tau_2 < \tau_2'}{(\tau_1 \to \tau_2) < (\tau_1' \to \tau_2')} \tag{12.8}$$

Ezt úgy mondjuk, hogy a függvény típus kontravariáns az első paraméterében, és kovariáns a második paraméterében.

Szorzat és összeg típusok kovariánsak minden paraméterükben (ez igaz az általános változataikra is).

$$\frac{\tau_1 < \tau_1' \qquad \tau_2 < \tau_2'}{\tau_1 \times \tau_2 < \tau_1' \times \tau_2'} \tag{12.9}$$

$$\frac{\tau_1 < \tau_1' \qquad \tau_2 < \tau_2'}{\tau_1 + \tau_2 < \tau_1' + \tau_2'} \tag{12.10}$$

A típusozás unicitása már nem igaz, de a haladás és a típusmegőrzés igaz. A típusozás inverzióját úgy kell megfogalmazni, hogy figyelembe vesszük az altípusokat.

13. Megoldások

4.1. feladat megoldása.

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

4.2. feladat megoldása.

$$
\begin{aligned}
&xs \in \mathsf{List}_{\mathsf{Nat}} \;::=\; \mathsf{nil} \mid \mathsf{cons}\,n\,xs \\
&\mathsf{nil}\text{ aritása} \quad ()\mathsf{List}_{\mathsf{Nat}} \\
&\mathsf{cons}\text{ aritása } (\mathsf{Nat}, \mathsf{List}_{\mathsf{Nat}})\mathsf{List}_{\mathsf{Nat}}
\end{aligned}
$$

4.3. feladat megoldása. Nyilak helyett a kötések alá egy sorszámot írtunk, és a kötött változók alá írtuk a kötés sorszámát.

$$
\begin{aligned}
&\overline{x} + \mathsf{num}\,\overline{i} \\
&\mathsf{let}\,\overline{x}\,\mathsf{in}\,\underset{1}{x}.\mathsf{num}\,\overline{i} + \underset{1}{\underline{x}} \\
&\mathsf{let}\,\overline{x}\,\mathsf{in}\,\underset{1}{x}.\overline{x'} + \mathsf{let}\,\underset{1}{\underline{x}}\,\mathsf{in}\,\underset{2}{x'}.\underset{2}{\underline{x'}} + \underset{1}{\underline{x}} \\
&\mathsf{let}\,\overline{x}\,\mathsf{in}\,\underset{1}{x}.\underset{1}{\underline{x}} + \mathsf{let}\,\underset{1}{\underline{x}}\,\mathsf{in}\,\underset{2}{x}.\underset{2}{\underline{x}} + \underset{2}{\underline{x}} \\
&\mathsf{let}\,\overline{x}\,\mathsf{in}\,\underset{1}{x}.(\mathsf{let}\,\underset{1}{\underline{x}}\,\mathsf{in}\,\underset{2}{x'}.\overline{x''} + \underset{2}{\underline{x'}}) + \mathsf{let}\,\overline{x'}\,\mathsf{in}\,\underset{3}{x'}.\underset{3}{\underline{x'}} + \overline{x''}
\end{aligned}
$$

4.4. feladat megoldása. Nem, mivel a let csak Exp fajtájú változót köt.

4.5. feladat megoldása.

$$
\begin{aligned}
x &\stackrel{?}{=} x' &&:= \text{igaz, ha } x = x' \\
& && \quad\;\; \text{hamis, egyébként} \\
x &\stackrel{?}{=} \mathsf{f}\,x'.t'\,t'_1 \dots t'_n &&:= \text{hamis} \\
\mathsf{f}\,x.t\,t_1 \dots t_n &\stackrel{?}{=} x &&:= \text{hamis} \\
\mathsf{f}\,x.t\,t_1 \dots t_n &\stackrel{?}{=} \mathsf{f}\,x'.t'\,t'_1 \dots t'_n &&:= \text{igaz, ha } t_1 = t'_1 \text{ és } \dots \text{ és } t_n = t'_n \text{ és } t[x \mapsto y] = t'[x' \mapsto y] \text{, ahol } y \text{ friss} \\
& && \quad\;\; \text{hamis, egyébként} \\
\mathsf{f}\,x.t\,t_1 \dots t_n &\stackrel{?}{=} \mathsf{g}\,x'.t'\,t_1 \dots t_m &&:= \text{hamis}
\end{aligned}
$$

4.6. feladat megoldása.

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

4.7. Nem, igen, nem, nem, igen, igen.

4.8 feladat megoldása. Példák:

$$\mathsf{d}\, y.y$$
$$\mathsf{d}\, y.\mathsf{g}\, y\, y$$
$$\mathsf{d}\, y.\mathsf{d}\, x.\mathsf{g}\, x\, y$$
$$\mathsf{d}\, y.\mathsf{g}\,(\mathsf{d}\, x.x)\, y$$

4.11 feladat megoldása.

$$\frac{}{\mathsf{max}\,\mathsf{zero}\, n\ \mathsf{is}\ n}$$

$$\frac{}{\mathsf{max}\, n\,\mathsf{zero}\ \mathsf{is}\ n}$$

$$\frac{\mathsf{max}\, n\, n'\ \mathsf{is}\ n''}{\mathsf{max}\,(\mathsf{suc}\, n)\,(\mathsf{suc}\, n')\ \mathsf{is}\ \mathsf{suc}\, n''}$$

4.12 feladat megoldása.

$$\frac{}{x\ \mathsf{hasHeight}\ \mathsf{zero}}$$

$$\frac{}{\mathsf{num}\, n\ \mathsf{hasHeight}\ \mathsf{zero}}$$

$$\frac{t\ \mathsf{hasHeight}\ n \qquad t'\ \mathsf{hasHeight}\ n' \qquad \mathsf{max}\, n\, n'\ \mathsf{is}\ n''}{t + t'\ \mathsf{hasHeight}\ \mathsf{suc}\, n''}$$

4.13 feladat megoldása.

$$\frac{}{\mathsf{isEven}\,\mathsf{zero}}$$

$$\frac{\mathsf{isEven}\, n}{\mathsf{isOdd}\,\mathsf{suc}\, n}$$

$$\frac{\mathsf{isOdd}\, n}{\mathsf{isEven}\,\mathsf{suc}\, n}$$

Pl. a következőképp vezetjük le, hogy a 3 páratlan.

$$\cfrac{\cfrac{\cfrac{\mathsf{isEven}\,\mathsf{zero}}{\mathsf{isOdd}\,\mathsf{suc}\,\mathsf{zero}}}{\mathsf{isEven}\,\big(\mathsf{suc}\,(\mathsf{suc}\,\mathsf{zero})\big)}}{\mathsf{isOdd}\,\Big(\mathsf{suc}\,\big(\mathsf{suc}\,(\mathsf{suc}\,\mathsf{zero})\big)\Big)}$$

4.14 feladat megoldása.
Igaz.

Bizonyítás. Az Exp fajtájú AST-ken szerkezeti indukcióval bizonyítjuk. $P(n) =$ = bármely n'-re létezik olyan n'', hogy $n+n'$ is n'' levezethető. n háromféle lehet (lásd 4.1. definíciót).

- Változó, ebben az esetben nem zárt, tehát nem felel meg a feladat feltételeinek, ezért nem kell bizonyítanunk semmit.

- $n = \mathsf{zero}$, ekkor $P(\mathsf{zero})$-t kell belátnunk, tehát azt, hogy bármely n'-re létezik olyan n'', hogy $\mathsf{zero} + n'$ is n'' levezethető. n''-t n'-nek választjuk, és alkalmazzuk a 4.3. levezetési szabályt.

- $n = \mathsf{suc}\, n_1$, ekkor azt kell belátnunk, hogy ha $P(n_1)$ igaz, akkor $P(\mathsf{suc}\, n_1)$ is igaz. $P(n_1)$ azt mondja, hogy minden n'-höz létezik olyan n_2, hogy $n_1 + n'$ is n_2 levezethető. $P(\mathsf{suc}\, n_1)$ bizonyításához egy tetszőleges n'-höz keresnünk kell egy olyan n''-t, melyre $\mathsf{suc}\, n_1 + n'$ is n'' levezethető. n''-t a $P(n_1)$ bizonyítása által megadott n_2-t felhasználva $\mathsf{suc}\, n_2$-nek választjuk. Ekkor, mivel tudjuk, hogy $n_1 + n'$ is n_2 levezethető, a 4.4. levezetési szabállyal megkapjuk, hogy $\mathsf{suc}\, n_1 + n'$ is $\mathsf{suc}\, n_2$.

$\square$

5.1. feladat megoldása.

operátor	aritás
(egész szám beágyazása)	$(\mathbb{Z})\mathsf{Exp}$
”–”	(szöveg)Exp
$- + -$	$(\mathsf{Exp}, \mathsf{Exp})\mathsf{Exp}$
$- - -$	$(\mathsf{Exp}, \mathsf{Exp})\mathsf{Exp}$
$- \bullet -$	$(\mathsf{Exp}, \mathsf{Exp})\mathsf{Exp}$
$\lvert - \rvert$	$(\mathsf{Exp})\mathsf{Exp}$
$\mathsf{let} - \mathsf{in}\, x.-$	$(\mathsf{Exp}, \mathsf{Exp}.\mathsf{Exp})\mathsf{Exp}$

5.9. feladat megoldása.

- $\cdot \vdash \lvert\mathsf{"ab"} \bullet \mathsf{"cd"}\rvert + \lvert\mathsf{let}\,\mathsf{"e"}\,\mathsf{in}\, x.x + x\rvert : \mathsf{int}$ nem típusozható. Az 5.4. lemma alapján mindig csak egyféle levezetési szabályt alkalmazhatunk, amikor alulról felfele felépítjük a levezetést. A ? lépésnél azonban elakadunk.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot \vdash \mathsf{"ab"} : \mathsf{str}}\,5.11 \qquad \dfrac{}{\cdot \vdash \mathsf{"cd"} : \mathsf{str}}\,5.11}{\cdot \vdash \mathsf{"ab"} \bullet \mathsf{"cd"} : \mathsf{str}}\,5.14}{\cdot \vdash \lvert\mathsf{"ab"} \bullet \mathsf{"cd"}\rvert : \mathsf{int}}\,5.15 \qquad \dfrac{\dfrac{\dfrac{}{\cdot \vdash \mathsf{"e"} : \mathsf{str}}\,5.11 \qquad \dfrac{?}{\cdot, x : \mathsf{str} \vdash x + x : \mathsf{str}}}{\cdot \vdash \mathsf{let}\,\mathsf{"e"}\,\mathsf{in}\, x.x + x : \mathsf{str}}\,5.16}{\cdot \vdash \lvert\mathsf{let}\,\mathsf{"e"}\,\mathsf{in}\, x.x + x\rvert : \mathsf{int}}\,5.15}{\cdot \vdash \lvert\mathsf{"ab"} \bullet \mathsf{"cd"}\rvert + \lvert\mathsf{let}\,\mathsf{"e"}\,\mathsf{in}\, x.x + x\rvert : \mathsf{int}}\,5.12$$

- Típusozható:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot \vdash \mathsf{"ab"} : \mathsf{str}}\,5.11 \qquad \dfrac{}{\cdot \vdash \mathsf{"cd"} : \mathsf{str}}\,5.11}{\cdot \vdash \mathsf{"ab"} \bullet \mathsf{"cd"} : \mathsf{str}}\,5.14}{\cdot \vdash \lvert\mathsf{"ab"} \bullet \mathsf{"cd"}\rvert : \mathsf{int}}\,5.15 \qquad \dfrac{\dfrac{\dfrac{}{\cdot \vdash \mathsf{"e"} : \mathsf{str}}\,5.11 \qquad \dfrac{\dfrac{(x : \mathsf{str}) \in \cdot, x : \mathsf{str}}{\cdot, x : \mathsf{str} \vdash x : \mathsf{str}}\,5.9 \qquad \dfrac{(x : \mathsf{str}) \in \cdot, x : \mathsf{str}}{\cdot, x : \mathsf{str} \vdash x : \mathsf{str}}\,5.9}{\cdot, x : \mathsf{str} \vdash x \bullet x : \mathsf{str}}\,5.14}{\cdot \vdash \mathsf{let}\,\mathsf{"e"}\,\mathsf{in}\, x.x \bullet x : \mathsf{str}}\,5.16}{\cdot \vdash \lvert\mathsf{let}\,\mathsf{"e"}\,\mathsf{in}\, x.x \bullet x\rvert : \mathsf{int}}\,5.15}{\cdot \vdash \lvert\mathsf{"ab"} \bullet \mathsf{"cd"}\rvert + \lvert\mathsf{let}\,\mathsf{"e"}\,\mathsf{in}\, x.x \bullet x\rvert : \mathsf{int}}\,5.12$$

- $\mathsf{let}\,\lvert\mathsf{"ab"} \bullet \mathsf{"cd"}\rvert\,\mathsf{in}\, x.x + x$ típusozható.

- $\mathsf{let}\,\lvert\mathsf{"ab"} \bullet \mathsf{"cd"}\rvert\,\mathsf{in}\, x.x \bullet x$ nem típusozható.

- $\lvert(\lvert\mathsf{"aaa"}\rvert)\rvert$ nem típusozható.

5.10. feladat megoldása.
Az 5.4. lemma kiegészítése minden esetre. Tfh. $\Gamma \vdash e : \tau$. Ekkor:

- Ha $e = x$, akkor $(x : \tau) \in \Gamma$.
- Ha $e = n$, akkor $\tau = \mathsf{int}$.
- Ha $e = "s"$, akkor $\tau = \mathsf{str}$.
- Ha $e = e_1 + e_2$, akkor $\tau = \mathsf{int}$, $\Gamma \vdash e_1 : \mathsf{int}$ és $\Gamma \vdash e_2 : \mathsf{int}$.
- Ha $e = e_1 - e_2$, akkor $\tau = \mathsf{int}$, $\Gamma \vdash e_1 : \mathsf{int}$ és $\Gamma \vdash e_2 : \mathsf{int}$.
- Ha $e = e_1 \bullet e_2$, akkor $\tau = \mathsf{str}$, $\Gamma \vdash e_1 : \mathsf{str}$ és $\Gamma \vdash e_2 : \mathsf{str}$.
- Ha $e = |e'|$, akkor $\tau = \mathsf{int}$ és $\Gamma \vdash e' : \mathsf{str}$.
- Ha $e = \mathsf{let}\, e_1 \,\mathsf{in}\, x.e'$, akkor valamely τ_1-re $\Gamma \vdash e_1 : \tau_1$ és $\Gamma, x : \tau_1 \vdash e' : \tau$.

Bizonyítás. $\Gamma \vdash e : \tau$ szerinti indukció. Ha az 5.9. szabállyal vezettük le, akkor $e = x$ és az indukciós hipotézisből kapjuk, hogy $(x : \tau) \in \Gamma$. Más szabály nincs, amivel $e = x$-et vezettünk volna le. Hasonlóan az összes többi esetre: minden operátorra pontosan egy levezetési szabályunk van. □

5.11. feladat megoldása.
Azt bizonyítjuk, hogy ha $\Gamma \vdash e : \tau$, akkor $\Gamma\,\mathsf{wf}$.

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

- Ha az 5.9. szabályt használtuk, tudjuk, hogy $(x : \tau) \in \Gamma$, és ebből szeretnénk belátni, hogy $\Gamma\,\mathsf{wf}$. Ezt $(x : \tau) \in \Gamma$ levezetése szerinti indukcióval látjuk be.
  - Ha az 5.7. szabályt használtuk, akkor $\Gamma = \Gamma', x : \tau$, és a szabály feltételeiből kapjuk, hogy $\Gamma'\,\mathsf{wf}$ és $x \notin \mathsf{dom}(\Gamma)$. Ezekből az 5.6. szabállyal megkapjuk, hogy $\Gamma', x : \tau\,\mathsf{wf}$.
  - Ha az 5.8. szabályt használtuk, akkor $\Gamma = \Gamma', y : \tau'$ és $y \notin \mathsf{dom}(\Gamma)$. Az indukciós feltevésből pedig tudjuk, hogy $\Gamma'\,\mathsf{wf}$. Megint az 5.6. szabállyal kapjuk, hogy $\Gamma', y : \tau'\,\mathsf{wf}$.
- Ha az 5.10–5.11. szabályok valamelyikét használtuk, akkor a szabály feltételéből kapjuk, hogy $\Gamma\,\mathsf{wf}$.
- Ha az 5.12–5.16. szabály valamelyikét használtuk, akkor az (egyik) indukciós feltevésből megkapjuk, hogy $\Gamma\,\mathsf{wf}$.

□

5.12. feladat megoldása.
Ha $\bullet$ helyett is $+$ szimbólumot használnánk, akkor az 5.14. szabály helyett a következő lenne.

$$\frac{\Gamma \vdash e_1 : \mathsf{str} \qquad \Gamma \vdash e_2 : \mathsf{str}}{\Gamma \vdash e_1 + e_2 : \mathsf{str}}$$

Ekkor az 5.2–5.8. lemmák mind teljesülnének, kivéve az 5.4. lemmát, amit kissé módosítani kellene. A következőt mondaná: ha $\Gamma \vdash e : \tau$ és $e = e_1 + e_2$, akkor $\Gamma \vdash e_1 : \tau$ és $\Gamma \vdash e_2 : \tau$, anélkül, hogy tudnánk, hogy $\tau = \mathsf{int}$ vagy $\tau = \mathsf{str}$.

5.13. feladat megoldása.

$$
\begin{aligned}
&(1+1) - |\text{"}\mathtt{aa}\text{"} \bullet \text{"}\mathtt{bb}\text{"}| \\
5.25 \longmapsto\ & 2 - |\text{"}\mathtt{aa}\text{"} \bullet \text{"}\mathtt{bb}\text{"}| \\
5.26 \longmapsto\ & 2 - 4 \\
5.22 \longmapsto\ & -2
\end{aligned}
$$

5.14. feladat megoldása.

Érték szerinti paraméterátadás esetén a $\mathsf{let}\,(1+1)\,\mathsf{in}\,x.x$ kifejezés kiértékelése során először elvégezzük az $1+1$ összeadást, majd a helyettesítés következik:

$$
\frac{\dfrac{}{1+1 \longmapsto 2}\ 5.21}{\mathsf{let}\,(1+1)\,\mathsf{in}\,x.x \longmapsto \mathsf{let}\,2\,\mathsf{in}\,x.x}\ 5.28 \qquad \frac{\dfrac{}{2\ \mathsf{val}}\ 5.19}{\mathsf{let}\,2 \in x.x \longmapsto x[x \mapsto 2] = 2}\ 5.29
$$

Név szerinti paraméterátadás esetén először helyettesítünk, majd összeadunk:

$$
\frac{}{\mathsf{let}\,(1+1)\,\mathsf{in}\,x.x \longmapsto x[x \mapsto 1+1] = 1+1}\ 5.29 \qquad \frac{}{1+1 \longmapsto 2}\ 5.21
$$

5.15. feladat megoldása. $\mathsf{let}\,(1+1)\,\mathsf{in}\,x.x+x$.

5.16. feladat megoldása. $\mathsf{let}\,(1+1)\,\mathsf{in}\,x.1$.

5.17. feladat megoldása.

$x+(1+2)$ kiértékelése elakad, egyik szabályt sem tudjuk alkalmazni, az 5.25. szabályt azért nem, mert a feltételét nem tudjuk biztosítani.

$$
\frac{\dfrac{}{1+2=3}\ 5.21}{(1+2)+x \longmapsto 3+x}\ 5.25,
$$

de aztán a kiértékelés elakad, mert az 5.26. szabály második feltételét nem tudjuk biztosítani.

5.20. feladat megoldása.

Megmutatjuk, hogy bármely s_1, s_2-re létezik olyan e, hogy $|\text{"}s_1\text{"} \bullet \text{"}s_2\text{"}| \longmapsto^*$ $*\ e$ és $|\text{"}s_1\text{"}| + |\text{"}s_2\text{"}| \longmapsto^* e$. Ha s_1 hossza n_1, s_2 hossza pedig n_2, és n_1 és n_2 összege n, akkor e-t n-nek választjuk. Megjegyezzük, hogy $e = n_1 + n_2$ nem lenne jó választás. Megmutatjuk, hogy mindkét kifejezés n-re értékelődik.

$$
\frac{\dfrac{}{\text{"}s_1\text{"} \bullet \text{"}s_2\text{"} \longmapsto \text{"}s_1s_2\text{"}}\ 5.23}{|\text{"}s_1\text{"} \bullet \text{"}s_2\text{"}| \longmapsto |\text{"}s_1s_2\text{"}|}\ 5.27 \qquad \frac{s_1s_2 \text{ hossza } n}{|\text{"}s_1s_2\text{"}| \longmapsto n}\ 5.27
$$

$$
\frac{\dfrac{}{|\text{"}s_1\text{"}| \longmapsto n_1}\ 5.24}{|\text{"}s_1\text{"}| + |\text{"}s_2\text{"}| \longmapsto n_1 + |\text{"}s_2\text{"}|}\ 5.25 \qquad \frac{n_1\ \mathsf{val} \qquad \dfrac{}{|\text{"}s_2\text{"}| \longmapsto n_2}\ 5.24}{n_1 + |\text{"}s_2\text{"}| \longmapsto n_1 + n_2}\ 5.26 \qquad \frac{}{n_1+n_2 \longmapsto n}\ 5.21
$$

5.24. feladat.

További szabályok:

$$
\frac{e_1\ \mathsf{err}}{e_1 \circ e_2\ \mathsf{err}}\ \circ \in \{-, \bullet\}
$$

$$
\frac{e_1\ \mathsf{val} \qquad e_2\ \mathsf{err}}{e_1 \circ e_2\ \mathsf{err}}\ \circ \in \{-, \bullet\}
$$

$$
\frac{e\ \mathsf{err}}{|e|\ \mathsf{err}}
$$

$$\frac{e_1 \ \mathsf{err}}{\mathsf{let}\, e_1 \,\mathsf{in}\, x.e_2 \ \mathsf{err}}$$

$$\frac{[e_1 \ \mathsf{val}] \qquad e_2 \ \mathsf{err}}{\mathsf{let}\, e_1 \,\mathsf{in}\, x.e_2 \ \mathsf{err}}$$

5.25. feladat.

5.26. feladat megoldása.

Megjegyezzük, hogy nem vezetünk be egy új karakter típust itt, hanem a head operátor csak egy egy-hosszú szöveget hoz létre (ha tud). Szintaxis kiegészítése:

$$e \in \mathsf{Exp} ::= \ldots \,|\, \mathsf{head}\, e$$

Típusrendszer kiegészítése:

$$\frac{\Gamma \vdash e : \mathsf{str}}{\Gamma \vdash \mathsf{head}\, e : \mathsf{str}}$$

Operációs szemantika kiegészítése:

$$\frac{e \longmapsto e'}{\mathsf{head}\, e \longmapsto \mathsf{head}\, e'}$$

$$\frac{s \text{ első betűje } s_1}{\mathsf{head}\, ''s'' \longmapsto ''s_1''}$$

$$\frac{s \text{ üres szöveg}}{\mathsf{head}\, ''s'' \ \mathsf{err}}$$

6.1. feladat.

int, str, $\mathsf{int} \to \mathsf{int}$, $\mathsf{int} \to \mathsf{str}$, $\mathsf{str} \to \mathsf{str}$, $\mathsf{str} \to \mathsf{int}$

6.2. feladat.

$\mathsf{fun}^{\mathsf{str}} x.x \bullet (x \bullet x)$

6.3. feladat. Az unicitás (5.3. lemma) nem teljesül, hiszen pl. $\cdot \vdash \mathsf{fun}\, x.x : \mathsf{str} \to \mathsf{str}$ és $\cdot \vdash \mathsf{fun}\, x.x : \mathsf{int} \to \mathsf{int}$ is levezethető. A többi lemma ugyanúgy bizonyítható, mint a **??**. fejezetben.

6.4. feladat.

Végtelen számú ilyen létezik:

$$\begin{aligned}
&\mathsf{str}\\
&\mathsf{int}\\
&\mathsf{str} \to \mathsf{str}\\
&\mathsf{str} \to \mathsf{int}\\
&\mathsf{int} \to \mathsf{str}\\
&\mathsf{int} \to \mathsf{int}\\
&\mathsf{str} \to (\mathsf{str} \to \mathsf{str})\\
&\mathsf{str} \to (\mathsf{str} \to \mathsf{int})\\
&\mathsf{str} \to (\mathsf{int} \to \mathsf{str})\\
&\mathsf{str} \to (\mathsf{int} \to \mathsf{int})\\
&(\mathsf{str} \to \mathsf{str}) \to \mathsf{str}\\
&(\mathsf{str} \to \mathsf{str}) \to \mathsf{int}\\
&\mathsf{str} \to (\mathsf{str} \to (\mathsf{str} \to \mathsf{str}))\\
&\mathsf{str} \to (\mathsf{int} \to (\mathsf{str} \to (\mathsf{int} \to \mathsf{str})))\\
&(\mathsf{int} \to \mathsf{int}) \to (\mathsf{str} \to (\mathsf{int} \to \mathsf{str}))\\
&\ldots
\end{aligned}$$

6.5. feladat. $\lambda^{\mathsf{int}} x.\lambda^{\mathsf{str}} y.x + |y|$
6.6. feladat. $\lambda^{\mathsf{str}\to\mathsf{str}} y.\lambda^{\mathsf{str}} x.y\,(y\,(y\,x))$
6.8. feladat.
6.9. feladat.

– név szerinti paraméterátadás:

$$(\lambda^{\mathsf{int}} x.x{+}x)\,(1{+}1) \xmapsto{6.17} (1{+}1){+}(1{+}1) \xmapsto{5.25,5.21} 2{+}(1{+}1) \xmapsto{5.26,5.21} 2{+}2 \xmapsto{5.21} 4$$

– érték szerinti paraméterátadás:

$$(\lambda^{\mathsf{int}} x.x + x)\,(1 + 1) \xmapsto{6.16,5.21} (\lambda^{\mathsf{int}} x.x + x)\,2 \xmapsto{6.17} 2 + 2 \xmapsto{5.21} 4$$

6.10. feladat.
7.1. feladat.
7.2. feladat.

– érték szerint, mohón

$$\begin{aligned}
&\lambda^{\mathsf{int}} x.\langle x + 3, x + 4\rangle\,(1 + 2)\\
\xmapsto{6.16,5.21}\,&\lambda^{\mathsf{int}} x.\langle x + 3, x + 4\rangle\,3\\
\xmapsto{6.17}\,&\langle 3 + 3, 3 + 4\rangle\\
\xmapsto{7.9,5.21}\,&\langle 6, 3 + 4\rangle\\
\xmapsto{7.10,5.21}\,&\langle 6, 7\rangle
\end{aligned}$$

– név szerint, mohón

$$\begin{aligned}
&\lambda^{\mathsf{int}}x.\langle x+3, x+4\rangle\,(1+2)\\
\xmapsto{6.17}&\langle(1+2)+3,(1+2)+4\rangle\\
\xmapsto{7.9,5.25,5.21}&\langle 3+3,(1+2)+4\rangle\\
\xmapsto{7.9,5.21}&\langle 6,(1+2)+4\rangle\\
\xmapsto{7.10,5.255.21}&\langle 6,(1+2)+4\rangle\\
\xmapsto{7.10,5.21}&\langle 6,7\rangle
\end{aligned}$$

– érték szerint lustán

$$\begin{aligned}
&\lambda^{\mathsf{int}}x.\langle x+3, x+4\rangle\,(1+2)\\
\xmapsto{6.16,5.21}&\lambda^{\mathsf{int}}x.\langle x+3, x+4\rangle\,3\\
\xmapsto{6.17}&\langle 3+3,3+4\rangle
\end{aligned}$$

– név szerint lustán

$$\begin{aligned}
&\lambda^{\mathsf{int}}x.\langle x+3, x+4\rangle\,(1+2)\\
\xmapsto{6.17}&\langle(1+2)+3,(1+2)+4\rangle
\end{aligned}$$

7.3. feladat.
7.4. feladat.

$$\lambda^{\mathsf{int}\times(\mathsf{int}\times\mathsf{int})}x.\mathsf{proj}_1\,x+\mathsf{proj}_1\,(\mathsf{proj}_2\,x)+\mathsf{proj}_2\,(\mathsf{proj}_2\,x)$$

7.5. feladat.
7.6. feladat.
7.7. feladat.
7.8. feladat.
$\mathsf{Bool} := \top + \top$
$\mathsf{true} := \mathsf{inj}_1\,\mathsf{tt}$
$\mathsf{false} := \mathsf{inj}_2\,\mathsf{tt}$
if e then e_1 else e_2 := $\mathsf{case}\,e\,x_1.e_1\,x_2.e_2$
7.9. feladat.
7.10. feladat.
7.11. feladat.
7.12. feladat.
7.13. feladat.
7.14. feladat.
7.15. feladat.
7.16. feladat.
7.17. feladat.

8.1. feladat.

1. $\big(X \supset (Y \supset Z)\big) \supset \big(Y \supset X \supset Z\big)$ bizonyítása üres feltétellista mellett:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot,(X \supset (Y \supset Z)) \ni (X \supset (Y \supset Z))}\,(8.3)}{\cdot,(X \supset (Y \supset Z)),Y \ni (X \supset (Y \supset Z))}\,(8.4)}{\cdot,(X \supset (Y \supset Z)),Y,X \ni (X \supset (Y \supset Z))}\,(8.4)}{\cdot,(X \supset (Y \supset Z)),Y,X \vdash (X \supset (Y \supset Z))}\,(8.5) \quad \dfrac{\dfrac{}{\cdot,(X \supset (Y \supset Z)),Y,X \ni X}\,(8.3)}{\cdot,(X \supset (Y \supset Z)),Y,X \vdash X}\,(8.5)}{\cdot,(X \supset (Y \supset Z)),Y,X \vdash Y \supset Z}\,(8.15) \quad \dfrac{\dfrac{\dfrac{}{\cdot,(X \supset (Y \supset Z)),Y \ni Y}\,(8.3)}{\cdot,(X \supset (Y \supset Z)),Y,X \ni Y}\,(8.4)}{\cdot,(X \supset (Y \supset Z)),Y,X \vdash Y}\,(8.5)}{\dfrac{\dfrac{\dfrac{\cdot,(X \supset (Y \supset Z)),Y,X \vdash Z}{\cdot,(X \supset (Y \supset Z)),Y \vdash (X \supset Z)}\,(8.14)}{\cdot,(X \supset (Y \supset Z)) \vdash (Y \supset X \supset Z)}\,(8.14)}{\cdot \vdash (X \supset (Y \supset Z)) \supset (Y \supset X \supset Z)}\,(8.14)}\,(8.15)
$$

2. $X \leftrightarrow (X \wedge \top)$ bizonyítása üres feltétellista mellett:
   Megjegyzés: $A \leftrightarrow B := (A \supset B) \wedge (B \supset A)$ miatt a feladat tulajdonképpen:
   $(X \supset (X \wedge \top)) \wedge ((X \wedge \top) \supset X)$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot,X \ni X}\,(8.3)}{\cdot,X \vdash X}\,(8.5) \quad \dfrac{}{\cdot,X \vdash \top}\,(8.6)}{\cdot,X \vdash (X \wedge \top)}\,(8.8)}{\cdot \vdash X \supset (X \wedge \top)}\,(8.14) \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\cdot,(X \wedge \top) \ni X \wedge \top}\,(8.3)}{\cdot,(X \wedge \top) \vdash X \wedge \top}\,(8.5)}{\cdot,(X \wedge \top) \vdash X}\,(8.9)}{\cdot \vdash ((X \wedge \top) \supset X)}\,(8.14)}{\cdot \vdash (X \supset (X \wedge \top)) \wedge ((X \wedge \top) \supset X)}\,(8.8)
$$

8.2. feladat.
8.3. feladat.
8.4. feladat.
9.1. feladat.
9.2. feladat.
$(\lambda^{\mathsf{Nat}} y.(\lambda^{Nat} z.(\mathsf{rec}\ \mathsf{zero}\ x.\mathsf{plus}\ x\ z)))\ y$
9.3. feladat.

Bizonyítás. Teljes indukciós bizonyítást alkalmazunk.
Nézzük meg $\overline{0} = \mathsf{zero}$-ra:
$\mathsf{plus}\ \mathsf{zero}\ \mathsf{zero} \longmapsto^* \mathsf{rec}\ \mathsf{zero}\ x.\mathsf{suc}\ x\ \mathsf{zero} \longmapsto \mathsf{zero}$
Tegyünk fel, hogy $\overline{n}$-re igaz, tehát $\mathsf{plus}\ \overline{n}\ \mathsf{zero} \longmapsto^* \overline{n}$

$$\mathsf{plus}\ \overline{n+1}\ \mathsf{zero}$$

Nézzük meg $\overline{n+1}$-re:

$$
\begin{aligned}
&\longmapsto^* \mathsf{rec}\ \mathsf{zero}\ x.\mathsf{suc}\ x\ \overline{n+1}\\
&\longmapsto \mathsf{suc}(x)[x \mapsto \mathsf{rec}\ \mathsf{zero}\ x.\mathsf{suc} x\ \overline{n}]\\
&= \mathsf{suc}(\overline{n}) = \overline{n+1}
\end{aligned}
$$

□

9.4. feladat.
9.5. feladat.
10.1. feladat.

$$\dfrac{\dfrac{}{\alpha.\bot\ \mathsf{poly}}\,10.4 \qquad \dfrac{\dfrac{\dfrac{}{\alpha.\top\ \mathsf{poly}}\,10.2 \quad \dfrac{}{\alpha.\alpha\ \mathsf{poly}}\,10.1}{\alpha.\top\times\alpha\ \mathsf{poly}}\,10.3 \quad \dfrac{\dfrac{}{\alpha.\alpha\ \mathsf{poly}}\,10.1 \quad \dfrac{}{\alpha.\alpha\ \mathsf{poly}}\,10.1}{\alpha.\alpha\times\alpha\ \mathsf{poly}}\,10.3}{\alpha.(\top\times\alpha)+\alpha\times\alpha\ \mathsf{poly}}\,10.5}{\alpha.(\bot+((\top\times\alpha)+\alpha\times\alpha)\ \mathsf{poly}}\,10.5$$

10.2. feladat.
$\bot + ((\top \times (\mathsf{int}+\mathsf{str})) + (\mathsf{int}+\mathsf{str}) \times (\mathsf{int}+\mathsf{str}))$
10.3. feladat.
Ehhez a feladathoz a következő szabályok szükségesek:

$$\dfrac{}{\alpha.\mathsf{Bool}\ \mathsf{poly}}$$

illetve

$$\dfrac{}{\mathsf{map}^{\alpha.\mathsf{Bool}}\ (x'.e'')\ e \longmapsto e} \qquad (*)$$

megoldás:

$$\begin{aligned}
&\mathsf{map}^{\alpha.\top+(\mathsf{Bool}\times\alpha}\ (x'.\mathsf{suc}x')\ (\mathsf{inj}_2\langle\mathsf{true}, e\rangle)\\
&\overset{10.12}{\longmapsto} \mathsf{case}\ \mathsf{inj}_2\langle\mathsf{true}, e\rangle\ x_1.\mathsf{inj}_1(\mathsf{map}^{\alpha.\top}(x'.\mathsf{suc}\ x')\ x_1)\\
&\qquad\qquad x_2.\mathsf{inj}_2(\mathsf{map}^{\alpha.\mathsf{Bool}\times\alpha}(x'.\mathsf{suc}\ x')x_2)\\
&\overset{7.36}{\longmapsto} \mathsf{inj}_2(\mathsf{map}^{\alpha.\mathsf{Bool}\times\alpha}(x'.\mathsf{suc}\ x')\ \langle\mathsf{true}, e\rangle)\\
&\overset{10.10}{\longmapsto} \mathsf{inj}_2\big\langle\mathsf{map}^{\alpha.\mathsf{Bool}}(x'.\mathsf{suc}\ x')\ \mathsf{true}, \mathsf{map}^{\alpha.\alpha}(x'.\mathsf{suc}\ x')\ e\big\rangle\\
&\overset{(*)}{\longmapsto} \mathsf{inj}_2\big\langle\mathsf{true}, \mathsf{map}^{\alpha.\alpha}(x'.\mathsf{suc}\ x')\ e\big\rangle\\
&\overset{10.8}{\longmapsto} \mathsf{inj}_2\big\langle\mathsf{true}, \mathsf{suc}\ e\big\rangle
\end{aligned}$$

10.4. feladat.
10.5. feladat.
10.6. feladat.
10.7. feladat.
10.8. feladat.
10.9. feladat.
10.10. feladat.
10.11. feladat.
10.12. feladat.
10.13. feladat.
10.14. feladat.
10.15. feladat.
10.16. feladat.
10.17. feladat.
10.19. feladat.

Hivatkozások

[1] A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley series in computer science and information processing. Addison-Wesley Publishing Company, 1986.

[2] Zoltán Csörnyei. *Fordítóprogramok*. Az informatika alkalmazásai. Typotex, 2009.

[3] Zoltán Csörnyei. *Bevezetés a típusrendszerek elméletébe*. ELTE Eötvös Kiadó, 2012.

[4] Robert Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, New York, NY, USA, 2012.

[5] Donald E. Knuth. Backus normal form vs. backus naur form. *Commun. ACM*, 7(12):735–736, December 1964.

[6] Benjamin C. Pierce. *Types and Programming Languages*. The MIT Press, 1st edition, 2002.

[7] The Univalent Foundations Program. Homotopy type theory: Univalent foundations of mathematics. Technical report, Institute for Advanced Study, 2013.

[8] The Agda development team. Agda wiki, 2017.