Nyelvek típusrendszere
(jegyzet)

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

2017. december 21.

Lektorálta: Csörnyei Zoltán

Az ELTE tankönyv- és jegyzettámogatási pályázatán elnyert forrás
felhasználásával készült.

Tartalomjegyzék

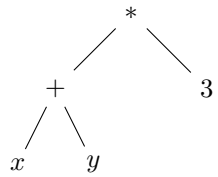
1. Bevezető

Egy fordítóprogram [1, 3] általában a következő részekből épül fel.

1. Lexikális elemző (lexer, scanner): karakterek listájából tokenek listáját készíti, például `"(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), például szükséges, hogy x és y numerikus típusú legyen. Ebben a jegyzetben ezzel a résszel foglalkozunk.

4. Kódgenerálás: ha a típusellenőrző átengedte a szintaxisfát, akkor ezt lefordítjuk a gép számára érthető kóddá.

Típusrendszerek és logika kapcsolatáról az alábbiakat jegyezzük meg.

– Mi az, hogy számítás? Erre egy válasz a lambda kalkulus (alternatíva a Turing-gépekre), amely a funkcionális programozási nyelvek (például Lisp, Haskell) alapja. A lambda kalkulust kiegészítették típusrendszerekkel, melyek kiszűrnek bizonyos programokat (lambda-kifejezéseket), melyek nem jól típusozhatók.

– Mi az, hogy bizonyítás? Erre válaszol a logika az ítéletlogikával, predikátumkalkulussal.

– Típuselmélet: egy olyan programozási nyelv (típusos lambda-kalkulus), mely egy tételbizonyító rendszer is, tehát az előbbi két pontot egyesíti. Ebben a jegyzetben a típuselmélet speciális eseteiről lesz szó. A típuselmélet használható a matematika megalapozására a halmazelmélet helyett. Előnye, hogy konstruktív (a bizonyítások egyben algoritmust is adnak), támogatja az absztrakciót (izomorf típusok nem különböztethetők meg), és egyszerűbb (például a függvények, a logikai implikáció és univerzális kvantor ugyanazzal a konstrukcióval van megadva).

> Típuselméletben programozva 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). Ebben a jegyzetben nem tárgyaljuk a típuselméletet. Az érdeklődőknek a homotópia típuselmélet könyv [10] első fejezeteit és az Agda programozási nyelvet [12] javasoljuk. [10].

A típusrendszerekről szól Csörnyei Zoltán könyve [4], a magyar elnevezésekben erre támaszkodunk. A tárgyalás során kisebb részben Pierce-t [8], nagyobb részben Harpert [5] követjük, néhány típusrendszer teljesen megegyezik [5] tárgyalásával.

Az új fogalmakat *dőlt betűvel* írjuk, mögötte általában felsoroljuk az alternatív- és az angol elnevezést, ezek a tárgymutatóban is megtalálhatók.

2. Szintaxis, típusrendszer, operációs szemantika

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 [6] adjuk meg.

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

Tm-et (term) *fajtának* (sort) nevezzük, ez egy halmaz, mely kifejezéseket tartalmaz. A $t, t', ...$-t *metaváltozóknak* (metavariable) nevezzük, mivel ezek a *metaelméletünk* (metatheory) változói. Metaelméletnek vagy metanyelvnek nevezzük azt a nyelvet, amiben a jegyzet mondatai, definíciói, bizonyításai íródnak. Ezt megkülönböztetjük az *objektumelmélettől* (objektumnyelvtől, object theory), 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 $\mathsf{num}\, 3 \in \mathsf{Tm}$. Ha t és t' termek, akkor $t + t'$ is egy term, például $(\mathsf{num}\, 0 + + \mathsf{num}\, 3) \in \mathsf{Tm}$, $(\mathsf{num}\, 3 + \mathsf{num}\, 0) \in \mathsf{Tm}$. 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 $(\mathsf{isZero}\,(\mathsf{num}\,2)) \in \mathsf{Tm}$ vagy $(\mathsf{isZero}\,(\mathsf{num}\,0 + + \mathsf{num}\,3)) \in \mathsf{Tm}$. 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:

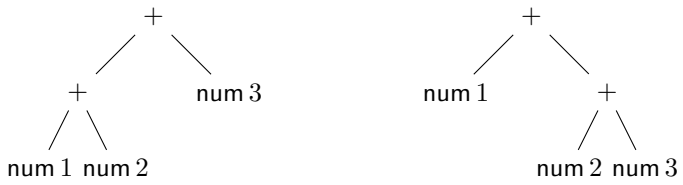
num	$(\mathbb{N})\mathsf{Tm}$
$- + -$	$(\mathsf{Tm}, \mathsf{Tm})\mathsf{Tm}$
isZero	$(\mathsf{Tm})\mathsf{Tm}$
true	$()\mathsf{Tm}$
false	$()\mathsf{Tm}$
$\mathsf{if} - \mathsf{then} - \mathsf{else} -$	$(\mathsf{Tm}, \mathsf{Tm}, \mathsf{Tm})\mathsf{Tm}$

$-$-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*, abstract syntax tree), 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 jegyzetben a szintaxissal ezen az absztakciós szinten foglalkozunk: nem foglalkozunk a szintaktikus elemzéssel [3] (parsing, hogyan lesz egy sztringből absztrakt szintaxisfa).

Nem minden termnek tudunk jelentést adni, például $\mathsf{isZero}\,\mathsf{true}$ és $\mathsf{true} + +\mathsf{num}\,3$ termek, de nem világos, hogy mit jelentenek. Az ilyen 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*

ezek aritása?

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

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

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

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

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

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

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

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

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

2.1.1. Term szerinti rekurzió és indukció

A fenti 2.1 definícióban bevezetett 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, illetve *indukcióval* (strukturális indukció) tudunk állításokat bizonyítani róluk.

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

adási módot rekurziónak).

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

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*-szal jelöljük a metaélméleti maximum függvényt.

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

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

Vegyük észre, hogy megint használtuk a metaelméleti összeadást.

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

Ha szeretnénk valamilyen állítást minden termről belátni, azt indukcióval tudjuk megtenni. Ez a természetes számokon alkalmazott teljes indukció általánosítása termekre. Ha egy $P(t)$ 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 adunk meg, 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 természetes 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

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

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

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

A $height(t) > height(t')$ eset hasonlóan belátható.

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

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

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

2.6. Feladat. *Készíts term-optimalizálást: ha van egy* $\mathsf{num}\,n + \mathsf{num}\,n'$ *részterm egy termben, ezt szeretnénk helyettesíteni* $\mathsf{num}\,(n+n')$*-vel, ahol* $+$ *a metaelméleti összeadás. Adj 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 11. fejezetben foglalkozunk velük.

2.2. Típusrendszer

A típusrendszer arra való, hogy kiszűrje a hibás kifejezéseket, az olyanokat, mint $\mathsf{isZero}\,\mathsf{true}$. A szűrést úgy végzi el, hogy a termeket különböző *típusokba* sorolja, és megadja, 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.

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

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 belátjuk Nat-ra és Bool-ra.

A *típusozási reláció* egy bináris reláció a termek és a típusok között, jelölése $- : - \subseteq \mathsf{Tm} \times \mathsf{Ty}$. Ha $t \in \mathsf{Tm}$ és $A \in \mathsf{Ty}$, akkor $t : A$-t *ítéletnek* (judgement) 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 definícióban leírt 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 2.3, 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* (derivable). A levezetési szabályokból *levezetési fákat* (levezetés, derivation trees) építünk fel. 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 írjuk le, 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? Add meg a levezetési fát, és jelöld, ha a levezetés elakad (nincs olyan szabály, ami alkalmazható)!*

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

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.

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

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.

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

Ezt a függvényt nevezzük a 2.1 definícióban megadott szintaxis *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, például 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. pontban 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. pont.

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:

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

– 2.5 szabály: ha $P(t : \mathsf{Nat})$, akkor $P(\mathsf{isZero}\, t : \mathsf{Bool})$,

– 2.6 szabály: $P(\mathsf{true} : \mathsf{Bool})$,

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

– 2.8 szabály: 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 módszert, a típusozás unicitását (tehát, hogy csak egyetlen típussal típusozható egy term) az alábbiakban belátjuk.

– 2.3 szabály. 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.

– 2.4 szabály. 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).

– Abból, hogy nem $t : \mathsf{Bool}$, nem $\mathsf{isZero}\, t : \mathsf{Nat}$, hasonló módon belátható.

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

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

– 2.8 szabály (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 sem 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 legfeljebb egyféleképpen típusozható (*típusok unicitása*).

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

Bizonyítás. Lásd a 2.2.1. pont 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

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

függvényt nevezünk, melyre $\mathit{infer}(t) = A \in \mathsf{Ty}$ pontosan akkor, ha $t : A$. A balról jobbra irányt *helyességnek* (soundness), a jobbról balra irányt *teljességnek* nevezzük (completeness).

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

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

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

Az odafele 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 szerint $\mathit{infer}(t'') = \mathsf{Nat}$ és

$\mathit{infer}(t') = \mathsf{Nat}$, az indukciós feltevésből emiatt megkapjuk, hogy $t'' : \mathsf{Nat}$ és $t' : \mathsf{Nat}$, és a 2.4 szabály alapján $t'' + t' : \mathsf{Nat}$.

– 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 szerint $\mathit{infer}(t') = \mathsf{Nat}$, ebből és az indukciós feltevésből megkapjuk, hogy $t' : \mathsf{Nat}$, és a 2.5 szabályt alkalmazva azt kapjuk, hogy $\mathsf{isZero}\, t' : \mathsf{Bool}$.

– A többi eset hasonló módon bizonyítható.

A visszafele irányt megkapjuk a $t : A$ 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 szerint 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 megkapjuk, hogy $\mathit{infer}(\mathsf{if}\, t'''\, \mathsf{then}\, t'\, \mathsf{else}\, t'') = A$.

□

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

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

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

$$\mathit{check}(t, A) := \begin{array}{ll} \mathtt{success}, & \textit{ha } \mathit{infer}(t) = A \\ \mathtt{fail} & \textit{, egyébként} \end{array}$$

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

2.3. Operációs szemantika

A szemantika a szintaxis jelentését adja meg. Korábban láttuk a Nat-Bool nyelv denotációs szemantikáját (2.2.1. pont), ebben csak a jól típusozott termeknek volt jelentése. Az operációs szemantika *kiértékeléssel* (interpretation, evaluation) adja meg a termek jelentését: azt mondja meg, hogy a term milyen másik termre értékelődik ki.

Ehhez először is meghatározzuk azokat a termeket, melyek *értékek* (values). Ehhez megadunk a termeken egy predikátumot (unáris relációt), melynek jelölése $- \mathsf{val} \subseteq \mathsf{Tm}$. Ezt induktívan, az alábbi levezetési szabályokkal adjuk meg.

$$\frac{}{\mathsf{num}\, n\, \mathsf{val}} \tag{2.9}$$

$$\frac{}{\mathsf{true}\, \mathsf{val}} \tag{2.10}$$

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

Ha $n \in \mathbb{N}$, akkor $\mathsf{num}\, n$ érték, ezenkívül true és false értékek.

A kiértékelést *átíró rendszerrel* (átíró rendszer, transition system) adjuk meg. Az átíró rendszer egyrészt az *egylépéses átírási relációból* áll, melynek jelölése $- \longmapsto - \subseteq \mathsf{Tm} \times \mathsf{Tm}$, és az $t \longmapsto t'$ ítélet azt fogja jelenteni, hogy t egy lépésben t'-re íródik át. Az átíró rendszer másik része a (nulla vagy több lépéses) *átírási reláció*, melynek jelölése $- \longmapsto^* - \subseteq \mathsf{Tm} \times \mathsf{Tm}$, és az egy lépéses átírási reláció reflexív tranzitív lezártjaként adjuk meg az alábbi levezetési szabályokkal.

$$\frac{}{e \longmapsto^* e} \tag{2.12}$$

$$\frac{e \longmapsto e' \qquad e' \longmapsto^* e''}{e \longmapsto^* e''} \tag{2.13}$$

A nulla vagy több lépéses átírási relációnak a későbbi fejezetekben is ugyanez lesz a definíciója, viszont az egylépéses átírási reláció változni fog a nyelvtől függően. A Nat-Bool nyelvre az egylépéses átírási relációt a következő levezetési szabályokkal adjuk meg.

$$\frac{n_1 \text{ és } n_2 \text{ összege } n}{\mathsf{num}\, n_1 + \mathsf{num}\, n_2 \longmapsto \mathsf{num}\, n} \tag{2.14}$$

$$\frac{}{\mathsf{isZero}\, (\mathsf{num}\, 0) \longmapsto \mathsf{true}} \tag{2.15}$$

$$\frac{n > 0}{\mathsf{isZero}\, (\mathsf{num}\, n) \longmapsto \mathsf{false}} \tag{2.16}$$

$$\frac{}{\mathsf{if\ true\ then}\ t_1\ \mathsf{else}\ t_2 \longmapsto t_1} \tag{2.17}$$

$$\frac{}{\mathsf{if\ false\ then}\ t_1\ \mathsf{else}\ t_2 \longmapsto t_2} \tag{2.18}$$

$$\frac{t_1 \longmapsto t_1'}{t_1 + t_2 \longmapsto t_1' + t_2} \tag{2.19}$$

$$\frac{t_1\ \mathsf{val} \qquad t_2 \longmapsto t_2'}{t_1 + t_2 \longmapsto t_1 + t_2'} \tag{2.20}$$

$$\frac{t \longmapsto t'}{\mathsf{isZero}\ t \longmapsto \mathsf{isZero}\ t'} \tag{2.21}$$

$$\frac{t \longmapsto t'}{\mathsf{if}\ t\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 \longmapsto \mathsf{if}\ t'\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2} \tag{2.22}$$

A fenti átírási szabályok azt adják meg, hogy a term hogyan fut, milyen lépésekben lesz belőle végül érték. Például a 2.14 szabály azt adja meg, hogy ha a $-+-$ operátort két $\mathsf{num}\,n$ alakú paraméterre alkalmazzuk, akkor az egy lépésben átíródik a két szám összegére (és ez már egy érték lesz). Hasonlóan, ha már tudjuk, hogy az isZero-nak megadott term egy szám, akkor át tudjuk írni igazra vagy hamisra, és ha a logikai elágazásnak true vagy false van megadva, átírjuk a megfelelő t_1 vagy t_2 termre.

A 2.14–2.18 szabályokat *utasítás szabályoknak* (instruction transitions) nevezzük, mert azt adják meg, hogy ha az operátor *fő paramétere* (principális paramétere) már ki van értékelve, akkor mi történjen. Például a $-+-$ operátornak mindkét paramétere fő paraméter, az isZero operátornak az egyetlen paramétere fő paraméter, míg az $\mathsf{if}-\mathsf{then}-\mathsf{else}-$ operátor első paramétere fő paraméter, a másik kettő nem: ha az első paraméter érték (true vagy false), akkor már tudjuk, hogy az első vagy a második ágra kell átírni az elágazást.

A 2.19–2.22 szabályokat *sorrendi szabályoknak* (search transitions, kongruencia szabályok) nevezzük, ezek adják meg, hogy milyen sorrendben értékelődjenek ki a fő paraméterek. Például a $-+-$ operátornak először az első paraméterét értékeljük ki (2.19), majd ha az első paramétere már egy érték, akkor értékeljük ki a második paraméterét (2.20).

Egy példa levezetés egy-lépéses átírásra:

$$\dfrac{\dfrac{}{\mathsf{num}\,1 + \mathsf{num}\,2 \longmapsto \mathsf{num}\,3}\ 2.14}{(\mathsf{num}\,1 + \mathsf{num}\,2) + (\mathsf{num}\,3 + \mathsf{num}\,4) \longmapsto \mathsf{num}\,3 + (\mathsf{num}\,3 + \mathsf{num}\,4)}\ 2.19$$

1 és 2 összege 3

Az eredményt pedig a következőképp írhatjuk át:

$$\dfrac{\dfrac{}{\mathsf{num}\,3\,\mathsf{val}}\ 2.9 \qquad \dfrac{3 \text{ és } 4 \text{ összege } 7}{\mathsf{num}\,3 + \mathsf{num}\,4 \longmapsto \mathsf{num}\,7}\ 2.14}{\mathsf{num}\,3 + (\mathsf{num}\,3 + \mathsf{num}\,4) \longmapsto \mathsf{num}\,3 + \mathsf{num}\,7}\ 2.20$$

Végül egy további lépésben eljutunk egy értékhez:

$$\dfrac{3 \text{ és } 7 \text{ összege } 10}{\mathsf{num}\,3 + \mathsf{num}\,7 \longmapsto \mathsf{num}\,10}\ 2.14$$

Ha az előbbi levezetéseketet elnevezzük r-nek, p-nek és q-nak, akkor több lépésben a következőhöz jutunk el:

$$\dfrac{\dfrac{\dfrac{}{\mathsf{num}\,1 + \mathsf{num}\,2) + (\mathsf{num}\,3 + \mathsf{num}\,4) \longmapsto^* \mathsf{num}\,10}\ 2.13}{\mathsf{num}\,3 + (\mathsf{num}\,3 + \mathsf{num}\,4) \longmapsto^* \mathsf{num}\,10}\ 2.13}{\mathsf{num}\,3 + \mathsf{num}\,7 \longmapsto^* \mathsf{num}\,10}\ 2.13}{\mathsf{num}\,10 \longmapsto^* \mathsf{num}\,10}\ 2.12$$

2.14. Feladat. *Értékeld ki az* $\mathsf{if}\,\mathsf{isZero}\,(\mathsf{num}\,1 + \mathsf{num}\,2)\,\mathsf{then}\,\mathsf{num}\,1\,\mathsf{else}\,(\mathsf{num}\,3 + + \mathsf{num}\,1)$ *kifejezést (írd le az egy-lépéses átírások levezetési fáit.)*

Bizonyos kifejezések kiértékelése *elakad* (gets stuck), ez azt jelenti, hogy a kifejezés nem érték, mégsincs olyan szabály, amellyel egy lépéssel tovább tudnánk haladni. Például a $\mathsf{true} + \mathsf{num}\,1$ kifejezés nem egy érték (nem vezethető le $\mathsf{true} + \mathsf{num}\,1\,\mathsf{val}$), és nem is írható át egy lépésben semmire, hiszen nem alkalmazható rá sem a 2.14 szabály, sem a 2.19 szabály (hiszen true-ra nincs átírási szabály), sem a 2.20 szabály (hiszen $\mathsf{num}\,1$-re sincs átírási szabály).

2.15. Feladat. *Írj még 3 olyan kifejezést, melyek kiértékelése elakadt!*

Vajon miért akad el ezen kifejezések kiértékelése?

2.16. Feladat. *Értékeld ki az* $\mathsf{if}\,\mathsf{true}\,\mathsf{then}\,\mathsf{num}\,1\,\mathsf{else}\,(\mathsf{num}\,1 + \mathsf{false})$ *kifejezést!*

2.17. Feladat. *Értékeld ki a* $(\mathsf{true} + \mathsf{true}) + (\mathsf{num}\,1 + \mathsf{num}\,2)$ *és az* $(\mathsf{num}\,1 + + \mathsf{num}\,2) + (\mathsf{true} + \mathsf{true})$ *kifejezéseket, és értelmezd a különbséget.*

2.18. Feladat. *Adj meg egy olyan* t' *kifejezést, melyben szerepel egy* t *metaváltozó, és ha* t*-t* t'*-ben* true*-ra helyettesítjük, akkor* $t' \longmapsto^* \mathsf{false}$*, ha pedig* false*-ra, akkor* $t' \longmapsto^* \mathsf{true}$*.*

A 2.18. feladat megoldásának t' kifejezése a logikai negáció műveletét modellezi. Magát a műveletet majd akkor tudjuk megadni, ha lesznek függvényeink a tárgyelméletünkben (lásd 5. fejezet).

2.19. Feladat. *Írd át a 2.19–2.20 szabályokat úgy, hogy a + először a második paraméterét értékelje ki, és csak utána az elsőt!*

Ha szeretnénk valamit bizonyítani az átíró rendszerünkről, az átírás levezetése szerinti indukciót alkalmazhatjuk rá. Ez azt jelenti, hogy ha $P(t \longmapsto t')$-t szeretnénk belátni minden $t \longmapsto t'$ levezetésre, akkor azt kell megmutatni, hogy a 2.14–2.22 szabályok megtartják P-t.

A következő tulajdonság azt fejezi ki, hogy nincs olyan átírási szabály, mely értéket írna át.

2.20. Lemma. *Nincs olyan t, hogy $t\,\mathsf{val}$ és $t \longmapsto t'$ valamely t'-re.*

Bizonyítás. $t \longmapsto t'$ szerinti indukció: egyik szabálykövetkezmény sem $\mathsf{num}\,n \longmapsto t'$, $\mathsf{true} \longmapsto t'$ vagy $\mathsf{false} \longmapsto t'$ alakú. □

A következő tulajdonságot *determináltságnak* nevezzük.

2.21. Lemma. *Ha $t \longmapsto t'$ és $t \longmapsto t''$, akkor $t' = t''$.*

Bizonyítás. $t \longmapsto t'$ levezetése szerinti indukció. Két esetet kiírunk.

- A 2.14 szabály használata esetén $t = \mathsf{num}\,n_1 + \mathsf{num}\,n_2$ és $t' = \mathsf{num}\,n$. A $t \longmapsto t''$ állítást csak a 2.14 szabállyal vezethettük le, mert még a 2.19 és a 2.20 szabályok következményében szerepel + a $\longmapsto$ bal oldalán, de ezeknek a szabályoknak a feltételében $\mathsf{num}\,n_1 \longmapsto t_1'$ illetve $\mathsf{num}\,n_2 \longmapsto t_2'$ szerepelne, ezek viszont nem levezethetők a 2.20. lemma miatt. Emiatt $t' = \mathsf{num}\,n = t''$.

- A 2.19 szabály esetén $t = t_1 + t_2$, $t' = t_1' + t_2$, és szeretnénk belátni, hogy ha $t_1 + t_2 \longmapsto t''$, akkor $t_1' + t_2 = t''$. Tudjuk azt is, hogy $t_1 \longmapsto t_1'$, emiatt a $t_1 + t_2 \longmapsto t''$ állítást csak a 2.19 szabállyal tudtuk levezetni, hiszen a 2.20 szabályhoz az kellene, hogy $t_1\,\mathsf{val}$, de a 2.20 lemma alapján ez nem lehet, a 2.14 szabályhoz pedig $t_1\,\mathsf{val}$ és $t_2\,\mathsf{val}$ lenne szükséges. Emiatt tudjuk, hogy $t'' = t_1'' + t_2$ valamely t_1''-re, és $t_1 \longmapsto t_1''$. Az indukciós hipotézisből tudjuk, hogy $t_1' = t_1''$, emiatt $t' = t_1' + t_2 = t_1'' + t_2 = t''$.

□

2.22. Feladat. *Alakítsd át a $-+$ $--$-ra vonatkozó sorrendi szabályokat úgy, hogy először a második paramétert értékelje ki, utána az elsőt! Módosítsd a 2.21. lemma bizonyítását, hogy igaz legyen erre is.*

2.23. Feladat. *Ha a 2.20 szabályból kihagyjuk a $t_1\,\mathsf{val}$ feltételt, be lehet-e bizonyítani a 2.21. lemmát? Ha igen, bizonyítsd be, ha nem, adj ellenpéldát!*

2.24. Feladat. *Továbbra is igazak maradnak-e a 2.20. és a 2.21. lemmák, ha a 2.19–2.20 szabályok helyett a következő szabályt adjuk meg?*

$$\frac{t_1 \longmapsto t_1' \qquad t_2 \longmapsto t_2'}{t_1 + t_2 \longmapsto t_1' + t_2'}$$

Ha igen, bizonyítsd be őket, ha nem, adj ellenpéldát!

2.25. Lemma. *Nincs végtelen hosszú átírási szekvencia, tehát minden t-re létezik olyan t', hogy $t \longmapsto^* t'$, de nincs olyan t'', hogy $t' \longmapsto t''$.*

Bizonyítás. t term szerinti indukcióval.

- Ha $t = \mathsf{num}\, n$, $t = \mathsf{true}$ vagy $t = \mathsf{false}$, akkor $t' := t$, és készen vagyunk.
- Ha $t = \mathsf{isZero}\, t_1$, akkor indukcióval t_1-re létezik ilyen t_1', és a 2.21 szabály iterációjával kapjuk, hogy $t \longmapsto^* \mathsf{isZero}\, t_1'$. Itt az alábbi eseteket különböztetjük meg:
  - Ha $t_1' = \mathsf{num}\, 0$, akkor $t' := \mathsf{true}$.
  - Ha $t_1' = \mathsf{num}\, n$, ahol $n > 0$, akkor $t' := \mathsf{false}$.
  - Egyébként $t' := \mathsf{isZero}\, t_1'$, hiszen nincs olyan szabály, amellyel ezt át tudnánk írni.
- A $t = t_1 + t_2$ eset hasonló.
- A $t = \mathsf{if}\, t_1\, \mathsf{then}\, t_2\, \mathsf{else}\, t_3$ esetben az indukciós feltevésből tudjuk, hogy létezik megfelelő tulajdonságú t_1', t_2', t_3'. Ha $t_1' = \mathsf{true}$, akkor $t' := t_2'$, ha $t_1' = \mathsf{false}$, akkor $t' := t_3'$, egyébként $t' := \mathsf{if}\, t_1'\, \mathsf{then}\, t_2\, \mathsf{else}\, t_3$.

□

2.26. Feladat. *Egészítsük ki az operációs szemantikát az alábbi szabállyal:*

$$\frac{}{\mathsf{num}\, 0 + t \longmapsto t}$$

Bizonyítsd be a 2.20–2.25. lemmákat az így kiegészített operációs szemantikára!

Mi történne, ha még az alábbi szabályt is hozzávennénk az operációs szemantikához?

$$\frac{}{t + \mathsf{num}\, 0 \longmapsto t}$$

2.27. Megjegyzés (A denotációs és operációs szemantika viszonyáról)**.** *A denotációs szemantikában (lásd 2.2.1. pont) a kiértékelés eredménye valamilyen metaelméleti strukturában van (például metaelméleti természetes szám), a kiértékelés sorrendjét szintén a metaelmélettől örököljük. Az operációs szemantika*

alacsonyabb szintű, a kiértékelés eredménye szintaxis, és a kiértékelés sorrendjét is mi határozzuk meg, például a + operátornak először az első, majd a második paraméterét értékeljük ki.

2.4. Típusrendszer és szemantika kapcsolata

Az operációs szemantikát a típusrendszer nélkül fogalmaztuk meg. Ebben a fejezetben azt nézzük meg, hogy mi a kettő kapcsolata. Ezt először az operátorok osztályozásával (bevezető- és eliminációs operátorok), majd két tétellel fogalmazzuk meg, a típusmegőrzéssel és a haladással.

A nyelvünk különböző típusokhoz tartozó operátorai kétféle csoportba oszthatók: *bevezető- és eliminációs operátorokra*. A Nat típusú termek bevezető operátora a num, eliminációs operátorai a $- + -$ és isZero. A Bool típusú termek bevezető operátorai a true és false és eliminátora az if – then – else –. A bevezető operátorokkal tudjuk létrehozni az adott típus elemeit, míg az eliminációs operátorokkal tudjuk felhasználni/lebontani őket. A – val ítélet szabályai azt mondják ki, hogy a bevezető operátorokkal megadott termek értékek. A 2.14–2.22 szabályok az eliminációs operátorokra adják meg, hogy azok hogyan viselkednek: a *sorrendi szabályok* azt mondják el, hogy melyek az eliminációs operátorok fő paraméterei, és milyen sorrendben kell őket kiértékelni, míg az *utasítás szabályok* azt adják meg, hogy mi történik, ha az eliminációs operátor összes fő paramétere bevezető operátor alakú.

A *típusmegőrzés* (tárgyredukció, subject reduction, type preservation) azt mondja ki, hogy ha van egy típusozható kifejezésünk, és egy átírási lépést végrehajtunk, ugyanazzal a típussal az átírt kifejezés az eredeti kifejezés típusával is típusozható.

2.28. Tétel. *Ha* $t : A$ *és* $t \longmapsto t'$*, akkor* $t' : A$*.*

Bizonyítás. $t \longmapsto t'$ szerinti indukció. Tehát $P(t \longmapsto t') =$ ha $t : A$, akkor $t' : A$. A következő eseteket kell megnéznünk:

- A 2.14 szabály esetén tudjuk, hogy $\mathsf{num}\, n_1 + \mathsf{num}\, n_2 : A$, ekkor a típuskikövetkeztetés (2.10. lemma) teljesség irányából következik, hogy $A = \mathsf{Nat}$, a 2.3 szabállyal pedig levezetjük, hogy $\mathsf{num}\, n : \mathsf{Nat}$.
- A 2.15 szabály esetén tudjuk, hogy $\mathsf{isZero}\,(\mathsf{num}\, 0) : A$, és a típuskikövetkeztetés teljesség irányából következik, hogy $A = \mathsf{Bool}$, és a 2.6 szabállyal pedig megkapjuk, hogy $\mathsf{true} : \mathsf{Bool}$.
- A 2.16 szabály esete hasonlóan bizonyítható.

– A 2.17 szabály esetén tudjuk, hogy $\mathsf{if}\,\mathsf{true}\,\mathsf{then}\,t_1\,\mathsf{else}\,t_2 : A$, és a típuskikövetkeztetés teljesség irányából következik, hogy $A = \mathit{infer}(t_1)$, és a helyesség irányból pedig következik, hogy $t_1 : \mathit{infer}(t_1)$.

– A 2.18 szabály esete hasonlóan bizonyítható.

– A 2.19 szabály esetén tudjuk, hogy $t_1 + t_2 : A$, a típuskikövetkeztetés teljességéből kapjuk, hogy $A = \mathsf{Nat}$, $\mathit{infer}(t_1) = \mathsf{Nat}$ és $\mathit{infer}(t_2) = \mathsf{Nat}$, és a helyességből kapjuk, hogy $t_1 : \mathsf{Nat}$ és $t_2 : \mathsf{Nat}$. Az indukciós hipotézisből és $t_1 : \mathsf{Nat}$-ból tudjuk, hogy $t_1' : \mathsf{Nat}$, emiatt a 2.4 alapján $t_1' + t_2 : \mathsf{Nat}$.

– A 2.20–2.22 szabályok esetei hasonlóan bizonyíthatók.

□

A *haladás* (progress) azt fejezi ki, hogy egy jól típusozott program végrehajtása nem akad el: vagy már maga egy érték, vagy még egy átírási lépést végre tudunk hajtani.

2.29. Tétel. *Ha $t : A$, akkor vagy $t\,\mathsf{val}$, vagy létezik olyan t', hogy $t \longmapsto t'$.*

A bizonyításhoz szükségünk van a következő lemmára.

2.30. Lemma. *Ha $t : A$ és $t\,\mathsf{val}$, akkor ha*

– *$A = \mathsf{Nat}$, akkor $t = \mathsf{num}\,n$ valamely n-re,*

– *$A = \mathsf{Bool}$, akkor $t = \mathsf{true}$ vagy $t = \mathsf{false}$.*

Bizonyítás. $t\,\mathsf{val}$ és $t : A$ szerinti indukció. □

A 2.29 tétel bizonyítása. $t : A$ levezetése szerinti indukció.

– A 2.3 szabály használata esetén $t = \mathsf{num}\,n$ és a 2.9 szabállyal kapjuk, hogy $\mathsf{num}\,n\,\mathsf{val}$.

– A 2.4 szabály használata esetén $t = t_1 + t_2$, ekkor az indukciós feltevés két esetet különböztet meg:

  - $t_1\,\mathsf{val}$, ebben az esetben a másik indukciós feltevés szerint két esetünk van:
    - $t_2\,\mathsf{val}$, ekkor a 2.30. lemma alapján $t_1 = \mathsf{num}\,n_1$ és $t_2 = \mathsf{num}\,n_2$ valamilyen n_1-re és n_2-re, és a 2.14 szabály alapján $t_1 + t_2 \longmapsto \mathsf{num}\,(n_1 + n_2)$.
    - Van olyan t_2', hogy $t_2 \longmapsto t_2'$, ekkor a 2.20 szabály alapján $t_1 + + t_2 \longmapsto t_1 + t_2'$.

- Van olyan t_1', hogy $t_1 \longmapsto t_1'$. Ekkor a 2.19 szabály alapján $t_1 + + t_2 \longmapsto t_1' + t_2$.

– A 2.5 szabály használata esetén $t = \mathsf{isZero}\, t_1$, ekkor az indukciós feltevés szerint két esetünk van:

- $t_1\, \mathsf{val}$, ekkor a 2.30. lemma alapján $t_1 = \mathsf{num}\, n$ valamely n-re. Ha $n = = 0$, akkor a 2.15 szabály alapján $\mathsf{isZero}\, t_1 \longmapsto \mathsf{true}$, egyébként a 2.16 szabály alapján $\mathsf{isZero}\, t_1 \longmapsto \mathsf{false}$.
- Van olyan t_1', hogy $t_1 \longmapsto t_1'$, ekkor a 2.21 szabály alapján $\mathsf{isZero}\, t_1 \longmapsto \mathsf{isZero}\, t_1'$.

– A 2.6 szabály használata esetén $t = \mathsf{true}$ és a 2.11 szabállyal kapjuk, hogy $\mathsf{true}\, \mathsf{val}$.

– A 2.7 szabály esete hasonlóan bizonyítható.

– A 2.8 szabály használata esetén $t = \mathsf{if}\, t_1\, \mathsf{then}\, t_2\, \mathsf{else}\, t_3$, ekkor az indukciós feltevésből t_1-re két esetünk van:

- $t_1\, \mathsf{val}$, ekkor a 2.30. lemma alapján $t_1 = \mathsf{true}$ vagy $t_1 = \mathsf{false}$, ekkor a 2.17 ill a 2.18 szabályok alapján lépünk tovább.
- Van olyan t_1', hogy $t_1 \longmapsto t_1'$, ekkor a 2.22 szabály alapján lépünk tovább.

□

Ha egy term típusozható, akkor nem akad el a kiértékelése. Igaz-e ez a másik irányban? A válasz: nem. Vannak olyan termek, melyek kiértékelhetők (néhány átírási lépés után értéket kapunk), mégsem típusozhatók.

2.31. Feladat. *Írjunk ilyen termet!*

Ez egy fontos dolgot elmond a típusrendszerről, ami általában is igaz: a típusrendszer megszorítja a programokat értelmes programokra, de mindig túl szigorú. Vannak olyan programok, melyek értelmesek, mégsem típusozhatók.

2.5. Futási idejű hibák

A Nat-Bool nyelvet kiegészíthetjük egy $-$ – $-$ kivonás operátorral az alábbi módon.

A szintaxishoz hozzávesszük az operátort.

$$t, t', ... \in \mathsf{Tm} ::= ... \,|\, t - t' \tag{2.23}$$

A típusrendszert kiegészítjük a következő szabállyal.

$$\frac{t_1 : \mathsf{Nat} \qquad t_2 : \mathsf{Nat}}{t_1 - t_2 : \mathsf{Nat}} \tag{2.24}$$

Az operációs szemantikát pedig az alábbi szabályokkal.

$$\frac{n_1 - n_2 = n}{\mathsf{num}\, n_1 - \mathsf{num}\, n_2 \longmapsto n} \tag{2.25}$$

$$\frac{t_1 \longmapsto t_1'}{t_1 - t_2 \longmapsto t_1' - t_2} \tag{2.26}$$

$$\frac{t_1\, \mathsf{val} \qquad t_2 \longmapsto t_2'}{t_1 - t_2 \longmapsto t_1 - t_2'} \tag{2.27}$$

A nyelvünk viszont elveszti a haladás tulajdonságát: van olyan term, ami nem érték, és nem is tudjuk átírni. Ilyen például a $\mathsf{num}\, 2 - \mathsf{num}\, 3$, hiszen nincs olyan $n \in \mathbb{N}$, hogy $2 - 3 = n$.

Ezt kétféleképpen lehet kezelni:

1. A típusozási szabállyal kizárjuk az ilyen eseteket: ehhez kell egy olyan típus, mely csak bizonyos számnál kisebb számokat tartalmaz, és akkor csak ilyen típusú értékeket engedünk kivonni a bizonyos számból. Ehhez függő típusrendszerre van szükségünk, lásd például [10].

2. Az ilyen típusú hibát futási időben ellenőrizzük, hibát ad a program futtatása.

Most az utóbbival foglalkozunk.

Bizonyos termek hibát okoznak, emiatt az operációs szemantikát egy új – err ítélettel és az alábbi szabállyal.

$$\frac{n_1 < n_2}{\mathsf{num}\, n_1 - \mathsf{num}\, n_2\, \mathsf{err}} \tag{2.28}$$

Ez azt mondja, hogy hibás állapotba kerülünk (futás közben), ha kisebb számból nagyobbat akarunk kivonni.

Továbbá meg kell mondanunk, hogy a különböző operátorok hogyan propagálják a hibákat.

$$\frac{t_1\, \mathsf{err}}{t_1 + t_2\, \mathsf{err}} \tag{2.29}$$

$$\frac{t_1\, \mathsf{val} \qquad t_2\, \mathsf{err}}{t_1 + t_2\, \mathsf{err}} \tag{2.30}$$

$$\frac{t\, \mathsf{err}}{\mathsf{isZero}\, t\, \mathsf{err}} \tag{2.31}$$

$$\frac{t\ \mathsf{err}}{\mathsf{if}\ t\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2\ \mathsf{err}} \tag{2.32}$$

$$\frac{t_1\ \mathsf{err}}{t_1 - t_2\ \mathsf{err}} \tag{2.33}$$

$$\frac{t_1\ \mathsf{val} \qquad t_2\ \mathsf{err}}{t_1 - t_2\ \mathsf{err}} \tag{2.34}$$

Ezen szabályok nélkül nem tudjuk, mit kellene kezdeni például az

$$\mathsf{if}\ \mathsf{isZero}\ (\mathsf{num}\ 2 - \mathsf{num}\ 3)\ \mathsf{then}\ \mathsf{false}\ \mathsf{else}\ \mathsf{true}$$

termmel.

Vegyük észre, hogy az if – then – else – operátornak csak a fő paraméterére adtuk meg a hibák propagálását: ha a többi paramétere hibás, akkor átírás után ugyanúgy hiba keletkezik.

A termeket kiegészíthetjük egy új, közvetlenül hibát megadó termmel.

$$t, t', ... \in \mathsf{Tm} ::= ... \mid \mathsf{error} \tag{2.35}$$

Ez a term tetszőleges típusú lehet:

$$\frac{}{\mathsf{error} : A}, \tag{2.36}$$

és az operációs szemantikája is hiba:

$$\frac{}{\mathsf{error}\ \mathsf{err}} \tag{2.37}$$

2.32. Feladat. *Egészítsük ki a 2.29. tételt hibákkal. Tehát bizonyítsuk be, hogy ha $t : A$, akkor vagy $t\ \mathsf{err}$, vagy $t\ \mathsf{val}$, vagy létezik olyan t', hogy $t \longmapsto t'$.*

2.33. Feladat. *Egészítsük ki a 2.28. tétel bizonyítását az új – — – és* error *operátorokkal.*

3. Lambda kalkulus

A lambda-kalkulust Church alkotta meg a 30-as években, eredetileg matematikai bizonyítások formalizálására. Turing megmutatta, hogy a lambda-kalkulusban leírható függvények pontosan a Turing-gépek által felismert függvények. Ebben a fejezetben bevezetjük a típus nélküli lambda kalkulust, és ezen keresztül a változó és változókötés fogalmát.

3.1. Szintaxis

Bevezetjük a *változók* Var fajtáját, ennek végtelen sok eleme van, melyeket x, y, z-vel és ezek vesszőzött, indexelt változataival jelölünk, és a Var fajtájú metaváltozókat is ugyanígy jelöljük. A változók egyenlősége eldönthető.

A lambda-kalkulus alapötlete, hogy a szokásos $f(x) = t$ jelölés helyett a függvényeket $\lambda x.t$-vel jelöljük (ahol t-ben előfordulhat x). Például $f(x) = x + 3$ helyett $\lambda x.x + 3$-at írunk λ-jelöléssel. Azt adjuk meg, hogy egy adott x bemenetre milyen kimenetet ad a függvény. A függvényalkalmazást (applikáció) pedig egymás után írással adjuk meg: $f(t')$ helyett $(\lambda x.t)\,t'$-t írunk.

A lambda-kalkulus termei háromfélék lehetnek: változó, absztrakció vagy applikáció.

$$t, t', ... \in \mathsf{Tm} ::= x \mid \lambda x.t \mid t\,t'$$

Az applikáció (melyet egymás után írással jelölünk) aritása $(\mathsf{Tm}, \mathsf{Tm})\mathsf{Tm}$. λ aritása $(\mathsf{Var}.\mathsf{Tm})\mathsf{Tm}$. λ-nak egy Tm fajtájú paramétere van, és ebben a paraméterben egy Var fajtájú változó van *kötve*, ezt jelöli a pont és az előtte lévő Var (a 10. fejezetben más fajtájú változókkal is fogunk találkozni).

Kötések előfordulnak a matematikában is: a $\sum_{x=0}^{10} 1/x$ kifejezésben a $\sum$ operátor köti az x változót az $1/x$ kifejezésben; hasonló a helyzet a $\lim_{x\mapsto\inf} 2^{-x}$ vagy a $\frac{dx^2+x+1}{dx}$ kifejezesékben. A programozásban a függvények definíciójában is kötéssel találkozunk: `int f(int i) { return(i+1); }`. Másik programozási példa a lokális definíciók, például `let x := 3+4 in x+2` (lásd 4. fejezet).

A kötés alapötlete, hogy a kötött változó neve közömbös: az előbbi szumma és $\sum_{y=0}^{10} 1/y$ megegyezik. Hasonlóan az `f` függvénnyel megegyezik az

```
int f(int j) { return(j+1); }
```

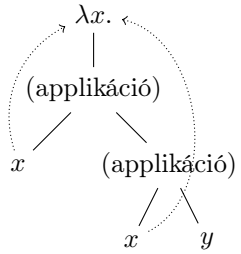
függvény.

A $\lambda x.t$ kifejezésben t-t a kötés *hatáskörének* (scope) nevezzük. Az x kötés hatáskörében szereplő x változókat *kötött változóknak* nevezzük (bound variables). Azokat a változókat, melyek nincsenek egy kötés hatáskörében, *szabad változóknak* nevezzük (free variables). A kötött változókat mutatóknak tekinthetjük, melyek a kötésre mutatnak, így a nevük közömbös: a $\lambda x.x\,x$ és a $\lambda y.y\,y$ kifejezések megegyeznek: ebben a jegyzetben ezeket a kifejezéseket egyenlőknek fogjuk

tekinteni, tehát nem is a 2.1. alfejezetben említett absztrakt szintaxisfa szinten tekintjük a kifejezéseket, hanem *absztrakt kötéses fa* szintjén (abstract binding tree). A típusrendszereket lehetne alacsonyabb absztrakciós szinten is tárgyalni, ilyenkor a $\lambda x.x\,x$ és a $\lambda y.y\,y$ kifejezések egyenlőségét α-ekvivalenciának nevezik.

Megjegyezzük, hogy a szabad változók neve természetesen nem közömbös, az $x\,(\lambda y.y)$ kifejezés és az $y\,(\lambda y.y)$ kifejezések különböznek.

A $\lambda x.(x\,(x\,y))$ kifejezésben például van két kötött és egy szabad változó. A kötéseket az alábbi szintaxisfán szaggatott nyíllal jelöljük.



Hogy ne kelljen túl sok zárójelet írni, az a konvenciónk, hogy a kötések hatásköre a lehető legtovább terjed, például $\lambda x.x\,(y\,z) = \lambda x.(x\,(y\,z))$, nem pedig $(\lambda x.x)\,(y\,z)$. Szintúgy $\lambda x.\lambda y.x\,y = \lambda x.(\lambda y.(x\,y))$. Az applikáció balra zárójeleződik, tehát $x\,y\,z = (x\,y)\,z$.

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

$$x\,y$$
$$x\,(\lambda x.y\,x)$$
$$x\,(\lambda x.x'\,(x\,(\lambda x'.x'\,x)))$$
$$x\,(\lambda x.x\,(x\,(\lambda x.x\,x)))$$
$$x\,(\lambda x.(x\,(\lambda x'.x''\,x')))\,(x'\,(\lambda x'.x'\,x''))$$

3.2. Feladat. *Döntsd el, hogy a következő termek megegyeznek-e!*

$$x\,y \stackrel{?}{=} x\,z$$
$$x\,y\,z \stackrel{?}{=} x\,(y\,z)$$
$$x\,(y\,z) \stackrel{?}{=} (x\,y)\,z$$
$$x\,(\lambda x.y\,x) \stackrel{?}{=} x\,(\lambda x'.y\,x')$$
$$x\,(\lambda x.y\,x) \stackrel{?}{=} x\,(\lambda x.y\,x')$$
$$x\,(\lambda x.x'\,(x\,(\lambda x'.x'\,x))) \stackrel{?}{=} x\,(\lambda x'.x\,(x'\,(\lambda x.x\,x')))$$
$$x\,(\lambda x.x'\,(x\,(\lambda x'.x'\,x))) \stackrel{?}{=} x\,(\lambda x''.x'\,(x''\,(\lambda x.x\,x'')))$$
$$x\,(\lambda x.x\,(x\,(\lambda x.x\,x))) \stackrel{?}{=} x\,(\lambda x'.x'\,(x'\,(\lambda x'.x'\,x')))$$

3.1.1. Rekurzió és indukció absztrakt kötéses fákon

Ebben a jegyzetben minden definíciót és bizonyítást úgy adunk meg, hogy az megtartja a „kötött változó neve közömbös" alapelvet. Emiatt a szerkezeti rekurzió és indukció elvét kissé módosítani kell.

Term szerinti rekurzióval megadhatjuk például az egy termben szereplő szabad változókat a következő módon.

$$\begin{aligned} &\mathit{free} \in \mathsf{Tm} \to \mathcal{P}(\mathsf{Var}) \\ &\mathit{free}(x) \quad := \{x\} \\ &\mathit{free}(\lambda x.t) := \mathit{free}(t) \backslash \{x\} \\ &\mathit{free}(t\,t') \quad := \mathit{free}(t) \cup \mathit{free}(t') \end{aligned}$$

Például $\mathit{free}(\lambda x.x\,y) = \{y\}$, $\mathit{free}(\lambda x.\lambda y.x\,y) = \{\}$ és $\mathit{free}(\lambda x.(y\,(\lambda y.y\,z))) = \{y, z\}$.

Általánosságban a szerkezeti rekurzió lambda-termekre azt mondja, hogy a következő három komponens meghatároz egy $\mathsf{Tm} \to A$ függvényt.

- minden $x \in \mathsf{Var}$-ra egy A-beli elem
- minden A-ra és $x \in \mathsf{Var}$-ra egy A-beli elem, melyben nem szerepelhet szabadon x
- minden $A \times A$ párra egy A-beli elem

A λ operátorhoz, mely egy változót köt a term paraméterében, tartozik az a feltétel, hogy az eredményben nem szerepelhet a kötött változó szabadon. Ez a feltétel biztosítja, hogy a „kötött változó neve közömbös" alapelvet a szerkezeti

rekurzióval megadott függvények betartják [9]. A fenti *free* példa így tényleg megadható, hiszen $\mathit{free}(t)\backslash\{x\}$-ben nem szerepel x.

Ha ez a feltétel nem lenne ott, akkor például megadható lenne egy függvény, mely egy termre megadja a benne kötött változókat, és ezzel meg tudnánk különböztetni a $\lambda x.x$ és a $\lambda y.y$ termeket, melyek egyenlők.

Egy következő példa az *átnevezés* függvény definíciója. $t\{y \mapsto z\}$ adja meg azt a termet, melyet úgy kapunk, hogy t-ben y összes előfordulását z-re cseréljük.

$$
\begin{aligned}
&-\{- \mapsto -\} \in \mathsf{Tm} \times \mathsf{Var} \times \mathsf{Var} \to \mathsf{Tm} \\
&x\{y \mapsto z\} \qquad := \{x\}, \text{ ha } x \neq y \\
&\qquad\qquad\quad\ \ \{z\}, \text{ egyébként} \\
&(\lambda x.t)\{y \mapsto z\} := \lambda x.(t\{y \mapsto z\}), \text{ ha } x \neq y \\
&\qquad\qquad\quad\ \ \lambda z.(t\{y \mapsto z\}), \text{ egyébként} \\
&(t\,t')\{y \mapsto z\} \quad := (t\{y \mapsto z\})\,(t'\{y \mapsto z\})
\end{aligned}
$$

Vegyük észre, hogy x nem szerepel szabadon a λ-hoz tartozó jobb oldalakon.

Term szerinti indukcióval bizonyíthatunk egy állítást termeken. Csak olyan t-től függő $P(t)$ állítást bizonyíthatunk, melyben $P(\lambda x.t)$-ben nem szerepel szabadon x. Lambda-termekre az indukció azt mondja ki, hogy ha megadjuk az alábbi három komponenst, akkor minden t-re $P(t)$ teljesül.

- minden x-re $P(x)$
- minden x-re és t-re, ha $P(t)$, akkor $P(\lambda x.t)$
- minden t-re és t'-re, ha $P(t)$ és $P(t')$, akkor $P(t\,t')$

Egy állítás például

$$P(t) = \text{ha } y \neq z \text{ és } y \in \mathit{free}(t), \text{ akkor } (t\{y \mapsto z\}) \neq t.$$

Ezt a következőképp bizonyítjuk minden t-re.

- Ha $t = x$, akkor $\mathit{free}(t) = \{x\}$, emiatt $x = y$, és így
$$x\{y \mapsto z\} = x\{x \mapsto z\} = z \neq y = x.$$
- Ha $t = t_1\,t_2$, akkor mivel $\mathit{free}(t_1\,t_2) = \mathit{free}(t_1) \cup \mathit{free}(t_2)$, $y \in \mathit{free}(t_1)$ vagy $y \in \mathit{free}(t_2)$. Az első esetet megnézzük, a másik szimmetrikus. $P(t_1)$ indukciós feltevésből tudjuk, hogy $t_1\{y \mapsto z\} \neq t_1$. Ha $(t_1\,t_2)\{y \mapsto z\} = = (t_1\{y \mapsto z\})\,(t_2\{y \mapsto z\}) = t_1\,t_2$, akkor ebből következik, hogy $t_1\{y \mapsto z\} \neq t_1$, ami ellentmondás.

– Ha $t = \lambda x.t_1$, akkor $\mathit{free}(\lambda x.t_1) = \mathit{free}(t_1)\backslash\{x\}$, emiatt $x \neq y$ és $y \in$ $\in \mathit{free}(t_1)$. Mivel $x \neq y$, $(\lambda x.t_1)\{y \mapsto z\} = \lambda x.(t_1\{y \mapsto z\})$, a $P(t_1)$ indukciós hipotézisből pedig $t_1\{y \mapsto z\} \neq t_1$, emiatt

$$\lambda x.t_1\{y \mapsto z\} \neq \lambda x.t_1.$$

3.2. Operációs szemantika

A változókat *helyettesítéssel* (substitution) tüntethetjük el (konkretizálhatjuk). $t[x \longmapsto t']$-el jelöljük azt a termet, melyet úgy kapunk, hogy t-ben x minden szabad előfordulását t'-vel helyettesítjük. Például:

$$(x\,(x\,y))[x \mapsto z\,z] = (z\,z)\,((z\,z)\,y)$$

Kötések alatt is helyettesítünk:

$$(\lambda y.x\,(x\,y))[x \mapsto z\,z] = \lambda y.(z\,z)\,((z\,z)\,y)$$

Ha a λ ugyanazt a változót köti, mint amit helyettesítünk, akkor *elfedés* (capture) valósul meg, és nem történik helyettesítés, például az alábbi esetben.

$$(\lambda x.x\,y)[x \mapsto z] = \lambda x.x\,y$$

Azonban

$$(x\,(\lambda x.x\,y))[x \mapsto z] = z\,(\lambda x.x\,y).$$

A kötések alatti helyettesítéssel vigyázni kell. Ha naiv módon helyettesítünk, akkor nem tartjuk be azt az alapelvet, hogy a kötött változók neve közömbös. Például ha $(\lambda x.x\,y)[y \mapsto x] = \lambda x.x\,x$, akkor mivel $\lambda x.x\,y = \lambda x'.x'\,y$, azt kapjuk, hogy $(\lambda x.x\,y)[y \mapsto x] = (\lambda x'.x'\,y)[y \mapsto x] = \lambda x'.x'\,x$, de ez nem egyenlő $\lambda x.x\,x$-vel. Ezt úgy oldjuk meg, hogy ha egy kötés alatt helyettesítünk, akkor a biztonság kedvéért átnevezzük a kötött változót valamilyen friss (még sehol sem használt) változóra (capture avoiding substitution).

Így a fenti szintaxisra a helyettesítést a következőképp adjuk meg:

$$\begin{aligned}
x[x \mapsto t'] &:= t' \\
y[x \mapsto t'] &:= y, \text{ ha } x \neq y \\
(t_1\,t_2)[x \mapsto t'] &:= (t_1[x \mapsto t'])\,(t_2[x \mapsto t']) \\
(\lambda y.t_1)[x \mapsto t'] &:= \lambda z.((t_1\{y \mapsto z\})[x \mapsto t']), \text{ ha } x \neq y \text{ és } z \text{ egy friss változó}
\end{aligned}$$

A változó akkor változik, ha épp őt helyettesítjük. Az applikáció helyettesítése

esetén a két paraméterben helyettesítünk, majd ezekre alkalmazzuk az applikáció operátort. λ esetén átnevezzük a kötött változót egy friss változóra, és ezután végezzük el a helyettesítést a kötés hatáskörében (vegyük észre, hogy a kötött változó átnevezése kezeli az elfedés esetét is).

A helyettesítést a későbbi nyelvek szintaxisaira nem fogjuk külön megadni, mert analóg módon történik: olyan paramétereknél, melyekben nincs kötés, rekurzívan helyettesítünk, olyan paramétereknél, melyekben van (egy vagy több) kötés, először átnevezzük a kötött változókat friss változókra, majd helyettesítünk a paraméterekben.

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

$$(x\,(\lambda x.x\,x))[x \mapsto z]$$
$$(x'\,(\lambda x.x'\,x))[x' \mapsto z]$$
$$(x'\,(\lambda x.x'\,x'))[x' \mapsto x]$$
$$(x'\,(\lambda x.x'\,x))[x' \mapsto x]$$

3.4. Feladat. *Bizonyítsd be, hogy ha $x \notin free(t)$, akkor $t[x \longmapsto t'] = t$.*

A lambda-kalkusuhoz nem adjuk meg, hogy mik az értékek, hanem csak egy utasítás szabályt adunk meg (β-szabály), mely a következő.

$$\frac{}{(\lambda x.t)\,t_1 \longmapsto t[x \mapsto t_1]} \tag{3.1}$$

Ez a szabály azt mondja, hogy ha egy függvényt, mely x bemenethez t kimenetet rendel, alkalmazunk egy t_1 bemenetre, akkor kimenetként t-t kapjuk, amelyben x összes előfordulása le van cserélve t_1-re. Például: $(\lambda x.y\,x\,x)\,(z\,z') \longmapsto y\,(z\,z')\,(z\,z')$.

A sorrendi szabályok azt mondják, hogy bárhol, bármilyen sorrendben végrehajtható az átírás.

$$\frac{t \longmapsto t'}{\lambda x.t \longmapsto \lambda x.t'} \tag{3.2}$$

$$\frac{t_1 \longmapsto t_1'}{t_1\,t_2 \longmapsto t_1'\,t_2} \tag{3.3}$$

$$\frac{t_2 \longmapsto t_2'}{t_1\,t_2 \longmapsto t_1\,t_2'} \tag{3.4}$$

Erre az operációs szemantikára természetesen nem igaz, hogy determinisztikus lenne (vö. 2.21. lemma).

Habár csak függvények vannak a lambda-kalkulusban, modellezni tudjuk a logikai értékeket a következő rövidítések bevezetésével (ezt *Church-kódolásnak*

(Church-encoding) hívjuk).

$$\begin{aligned} \mathit{true} &:= \lambda x.\lambda y.x \\ \mathit{false} &:= \lambda x.\lambda y.y \\ \mathit{ifthenelse} &:= \lambda x.x \\ \mathit{and} &:= \lambda x.\lambda y.x\, y\, \mathit{false} \end{aligned}$$

3.5. Lemma. *Az így megadott ifthenelse úgy működik, ahogy elvárjuk, tehát:*

$$\begin{aligned} \mathit{ifthenelse}\ \mathit{true}\, t_1\, t_2 &\longmapsto^* t_1, \\ \mathit{ifthenelse}\ \mathit{false}\, t_1\, t_2 &\longmapsto^* t_2. \end{aligned}$$

Bizonyítás. Először is

$$\mathit{ifthenelse}\ \mathit{true}\, t_1\, t_2 = ((\lambda x.x)\,(\lambda x.\lambda y.x))\, t_1\, t_2.$$

Erre

$$\dfrac{\dfrac{\dfrac{}{(\lambda x.x)\,(\lambda x.\lambda y.x) \longmapsto x[x \mapsto \lambda x.\lambda y.x]}\ 3.1}{((\lambda x.x)\,(\lambda x.\lambda y.x))\, t_1 \longmapsto (x[x \mapsto \lambda x.\lambda y.x])\, t_1}\ 3.3}{((\lambda x.x)\,(\lambda x.\lambda y.x))\, t_1\, t_2 \longmapsto (x[x \mapsto \lambda x.\lambda y.x])\, t_1\, t_2}\ 3.3$$

és

$$(x[x \mapsto \lambda x.\lambda y.x])\, t_1\, t_2 = (\lambda x.\lambda y.x)\, t_1\, t_2.$$

Erre

$$\dfrac{\dfrac{}{(\lambda x.\lambda y.x)\, t_1 \longmapsto (\lambda y.x)[x \longmapsto t_1]}\ 3.1}{(\lambda x.\lambda y.x)\, t_1\, t_2 \longmapsto (\lambda y.x)[x \longmapsto t_1]\, t_2}\ 3.3$$

és

$$(\lambda y.x)[x \longmapsto t_1]\, t_2 = (\lambda z.t_1)\, t_2,$$

ahol z egy friss változó, tehát nem szerepel t_1-ben. Végül

$$\dfrac{}{(\lambda z.t_1)\, t_2 \longmapsto t_1[z \longmapsto t_2]}\ 3.1$$

és mivel z nem szerepel t_1-ben, a 3.4. feladat bizonyítása alapján

$$t_1[z \longmapsto t_2] = t_1.$$

A *false* eset bizonyítása hasonló. □

3.6. Feladat. *Mutasd meg, hogy az and úgy működik, ahogy elvárjuk, tehát*

$$\begin{aligned} \mathit{and\ true\ true} &\longmapsto^* \mathit{true},\\ \mathit{and\ true\ false} &\longmapsto^* \mathit{false},\\ \mathit{and\ false\ true} &\longmapsto^* \mathit{false},\\ \mathit{and\ false\ false} &\longmapsto^* \mathit{false}. \end{aligned}$$

3.7. Feladat. *Add meg a logikai tagadás és a diszjunkció operátorát.*

A természetes számok is reprezentálhatók Church-kódolással.

$$\begin{aligned} &\mathit{0} := \lambda x.\lambda y.y\\ &\mathit{1} := \lambda x.\lambda y.x\,y\\ &\mathit{2} := \lambda x.\lambda y.x\,(x\,y)\\ &\mathit{3} := \lambda x.\lambda y.x\,(x\,(x\,y))\\ &\mathit{stb.} \end{aligned}$$

A rákövetkező operátor a következő módon adható meg.

$$\mathit{suc} := \lambda z.\lambda x.\lambda y.x\,(z\,x\,y)$$

3.8. Feladat. *Mutasd meg, hogy suc 2* $\longmapsto^*$ *3.*

3.9. Feladat. *Add meg az összeadás operátort.*

3.10. Feladat. *Add meg az iszero operátort, melyre igaz, hogy iszero 0* $\longmapsto^*$ $*$ *true és minden t-re iszero* $(\mathit{suc}\,t) \longmapsto^*$ *false.*

3.11. Feladat. *Add meg a pred operátort, melyre, ha t* val *pred* $(\mathit{suc}\,t) \longmapsto^*$ *t! (Nehéz.)*

A lambda-kalkulusban nem igaz, hogy minden átírási szekvencia véges (vö. 2.25. lemma). Sőt, meg tudunk adni olyan termet, hogy $t \longmapsto t$:

$$\frac{}{(\lambda x.x\,x)\,(\lambda x.x.x) \longmapsto (\lambda x.x.x)\,(\lambda x.x.x)}\ 3.1$$

Mi több, tetszőleges rekurzívan megadott programot meg tudunk adni lambda-termként (ez az alapötlet Turing bizonyításában, mely megmutatja, hogy a lambda-kalkulussal és Turing-gépekkel ugyanazok a programok írhatók le). Ha van egy rekurzívan megadott függvényünk, például

$$f(n) = \mathit{ifthenelse}\,(\mathit{iszero}\,n)\ \mathit{1}\ (\mathit{2} + (f(\mathit{pred}\,n))),$$

ezt átírhatjuk a következő kifejezésre:

$$t := \lambda f.\lambda n.\mathit{ifthenelse}\,(\mathit{iszero}\,n)\;\mathit{1}\;(\mathit{2} + (f\,(\mathit{pred}\,n))).$$

Ekkor az f által reprezentált programot megkapjuk a következőképp:

$$t_1 := (\lambda x.t\,(x\,x))\,(\lambda x.t\,(x\,x)).$$

Hogy lássuk, miért működik ez pont úgy, mint f-től várjuk, adjunk meg egy t' paramétert, és hajtsunk végre néhány átírási lépést:

$$\begin{aligned}
& t_1\,t' \\
= \quad & (\lambda y.t\,(y\,y))\,(\lambda y.t\,(y\,y))\,t' \\
\longmapsto\ & t\,((\lambda y.t\,(y\,y))\,(\lambda y.t\,(y\,y)))\,t' \\
= \quad & t\,t_1\,t' \\
\longmapsto\ & (\lambda n.\mathit{ifthenelse}\,(\mathit{iszero}\,n)\;\mathit{1}\;(\mathit{2} + (t_1\,(\mathit{pred}\,n))))\,t' \\
\longmapsto\ & \mathit{ifthenelse}\,(\mathit{iszero}\,t')\;\mathit{1}\;(\mathit{2} + (t_1\,(\mathit{pred}\,t')))
\end{aligned}$$

Tehát pontosan egy lépés rekurziót hajtottunk végre, és a rekurzív függvényhívásnál megint t_1 található.

További információért lásd [2, 11].

4. Lokális definíciók

Ebben a fejezetben kiegészítjük a Nat-Bool nyelvet helyi (lokális) definíciókkal.

4.1. Szintaxis

A 2.1 definícióban megadott szintaxist kiegészítjük változókkal és a let operátorral.

$$t, t', ... \in \mathsf{Tm} ::= ... \mid x \mid \mathsf{let}\, t\, \mathsf{in}\, x.t' \tag{4.1}$$

A let operátorral tudunk helyi definíciókat írni, például

$$\mathsf{let}\, \mathsf{num}\, 2 + \mathsf{num}\, 3\, \mathsf{in}\, x.x + (x + x).$$

Ez azt fogja jelenteni (lásd 4.3. alfejezet), hogy elnevezzük $\mathsf{num}\, 2 + \mathsf{num}\, 3$-at x-nek, és $x + (x + x)$-ben x-et erre helyettesítjük. Tehát majd a jelentése ennek a termnek az lesz, hogy

$$(\mathsf{num}\, 2 + \mathsf{num}\, 3) + ((\mathsf{num}\, 2 + \mathsf{num}\, 3) + (\mathsf{num}\, 2 + \mathsf{num}\, 3)).$$

A let operátor aritása $(\mathsf{Tm}, \mathsf{Var}.\mathsf{Tm})\mathsf{Tm}$: a második paraméterében köt egy változót.

Alternatív jelölés a fenti példa kifejezésre a

```
let x:=num 2+num 3 in x+(x+x)
```

vagy az alábbi.

```
x+(x+x)
  where
    x:=num 2+num 3
```

4.2. Típusrendszer

A típusok nem változnak, továbbra is Nat és Bool a két típus. A típusrendszer levezetési szabályai viszont megváltoznak, mert figyelembe kell vennünk a változókat is. Ehhez bevezetünk egy új fajtát, a *környezetek* (context) fajtáját. Egy környezet változó-típus pároknak a listája.

$$\Gamma, \Gamma', ... \in \mathsf{Con} ::= \cdot \mid \Gamma, x : A \tag{4.2}$$

$\cdot$-tal jelöljük az üres környezetet (egy változó-típus pár sincs a környezetben), és ha már van egy Γ környezetünk, azt ki tudjuk egészíteni egy A típusú x változóval.

A célunk a környezettel az, hogy megadjuk a kifejezésekben levő szabad változók típusait. A típusozási ítélet $\Gamma \vdash t : A$ formájúra változik, ami azt mondja, hogy a t kifejezésnek a típusa A, feltéve, hogy a szabad változóinak típusai a Γ által adottak.

Hogy egyértelmű legyen, melyik változóhoz milyen típust rendelünk, bevezetünk egy megszorítást a környezetekre: egy változó csak egyszer szerepelhet. Először is megadunk egy függvényt, mely kiszámítja a környezet változóit tartalmazó halmazt.

$$\begin{aligned} dom(\cdot) &:= \{\} \\ dom(\Gamma, x : A) &:= \{x\} \cup dom(\Gamma) \end{aligned} \tag{4.3}$$

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

$$\frac{}{\cdot\,\mathsf{wf}} \tag{4.4}$$

$$\frac{\Gamma\,\mathsf{wf} \qquad x \notin dom(\Gamma)}{\Gamma, x : A\,\mathsf{wf}} \tag{4.5}$$

Az üres környezet jól formált, és ha van egy jól formált környezetünk, azt kiegészíthetjük egy új változó-típus párral, ha a változó még nem szerepelt a környezetben.

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

$$\frac{\Gamma\,\mathsf{wf} \qquad x \notin dom(\Gamma)}{(x : A) \in \Gamma, x : A} \tag{4.6}$$

$$\frac{(x : A) \in \Gamma \qquad y \notin dom(\Gamma)}{(x : A) \in \Gamma, y : A'} \tag{4.7}$$

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

A típusrendszerrel $\Gamma \vdash t : A$ formájú ítéleteket lehet levezetni, ami azt jelenti, hogy a Γ környezetben a t kifejezésnek A típusa van. Úgy is gondolhatunk erre, hogy t egy A típusú program, és a program paraméterei és azok típusai Γ-ban vannak megadva. A következő szabályokkal adjuk meg a típusrendszert.

Először megadjuk a 2.3–2.8 szabályok környezetekkel kiegészített változatait.

$$\frac{\Gamma\,\mathsf{wf}}{\Gamma \vdash \mathsf{num}\,n : \mathsf{Nat}} \tag{4.8}$$

$$\frac{\Gamma \vdash t : \mathsf{Nat} \qquad \Gamma \vdash t' : \mathsf{Nat}}{\Gamma \vdash t + t' : \mathsf{Nat}} \tag{4.9}$$

$$\frac{\Gamma \vdash t : \mathsf{Nat}}{\Gamma \vdash \mathsf{isZero}\, t : \mathsf{Bool}} \tag{4.10}$$

$$\frac{\Gamma\ \mathsf{wf}}{\Gamma \vdash \mathsf{true} : \mathsf{Bool}} \tag{4.11}$$

$$\frac{\Gamma\ \mathsf{wf}}{\Gamma \vdash \mathsf{false} : \mathsf{Bool}} \tag{4.12}$$

$$\frac{\Gamma \vdash t : \mathsf{Bool} \qquad \Gamma \vdash t' : A \qquad \Gamma \vdash t'' : A}{\Gamma \vdash \mathsf{if}\, t\, \mathsf{then}\, t'\, \mathsf{else}\, t'' : A} \tag{4.13}$$

Ha a környezetben van egy A típusú x változó, akkor az típusozható.

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A} \tag{4.14}$$

A lokális definíció típusozási szabálya a következő.

$$\frac{\Gamma \vdash t_1 : A_1 \qquad \Gamma, x : A_1 \vdash t_2 : A_2 \qquad x \notin dom(\Gamma)}{\Gamma \vdash \mathsf{let}\, t_1\, \mathsf{in}\, x.t_2 : A_2} \tag{4.15}$$

Példa levezetés:

$$\dfrac{\dfrac{\dfrac{}{\cdot\ \mathsf{wf}}\ 4.4}{\cdot \vdash \mathsf{num}\, 3 : \mathsf{Nat}}\ 4.8 \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot\ \mathsf{wf}}\ 4.4 \qquad x \notin \{\}}{(x : \mathsf{Nat}) \in \cdot, x : \mathsf{Nat}}\ 4.6}{\cdot, x : \mathsf{Nat} \vdash x : \mathsf{Nat}}\ 4.14 \qquad \dfrac{\dfrac{\dfrac{}{\cdot\ \mathsf{wf}}\ 4.4 \qquad x \notin \{\}}{\cdot, x : \mathsf{Nat}\ \mathsf{wf}}\ 4.5}{\cdot, x : \mathsf{Nat} \vdash \mathsf{num}\, 2 : \mathsf{Nat}}\ 4.8}{\cdot, x : \mathsf{Nat} \vdash x + \mathsf{num}\, 2 : \mathsf{Nat}}\ 4.9}{\cdot, x : \mathsf{Nat} \vdash \mathsf{isZero}\, (x + \mathsf{num}\, 2) : \mathsf{Bool}}\ 4.10}{\cdot \vdash \mathsf{let}\, \mathsf{num}\, 3\, \mathsf{in}\, x.\mathsf{isZero}\, (x + \mathsf{num}\, 2) : \mathsf{Bool}}\ 4.15$$

Erre a típusrendszerre is igazak a következő tulajdonságok, melyeket a Nat-Bool típusrendszerre láttunk.

4.1. Lemma. *A típusrendszer nem triviális (vö. 2.8. lemma).*

Bizonyítás. Például az üres környezetben a true+true term nem típusozható. □

Típusok *unicitása*:

4.2. Lemma. *Egy t termhez és Γ környezethez legfeljebb egy olyan A típus tartozik, melyre $\Gamma \vdash t : A$.*

Bizonyítás. Hasonló, mint a 2.9. lemma esetében. □

A *típuskikövetkeztetésnek* most egy környezetre is szüksége van, mely megadja a szabad változók típusát.

4.3. Lemma. *Az alábbi infer függvény kikövetkezteti a term típusát, ha* $\Gamma\,\mathsf{wf}$.

$$
\begin{array}{lll}
\mathit{infer}_{\mathsf{Con}} \in \mathsf{Con} \times \mathsf{Var} \to \mathsf{Ty} \cup \{\mathtt{fail}\} & & \\
\mathit{infer}_{\mathsf{Con}}(\cdot, x) & := \mathtt{fail} & \\
\mathit{infer}_{\mathsf{Con}}((\Gamma, x : A), y) & := A, & \textit{ha } x = y \\
\mathit{infer}_{\mathsf{Con}}((\Gamma, x : A), y) & := \mathit{infer}_{\mathsf{Con}}(\Gamma, y), & \textit{egyébként} \\
 & & \\
\mathit{infer} \in \mathsf{Con} \times \mathsf{Tm} \to \mathsf{Ty} \cup \{\mathtt{fail}\} & & \\
\mathit{infer}(\Gamma, \mathsf{num}\, n) & := \mathsf{Nat} & \\
\mathit{infer}(\Gamma, t + t') & := \mathsf{Nat}, & \textit{ha } \mathit{infer}(\Gamma, t) = \mathsf{Nat} \\
 & & \textit{és } \mathit{infer}(\Gamma, t') = \mathsf{Nat} \\
 & \mathtt{fail}, & \textit{egyébként} \\
\mathit{infer}(\Gamma, \mathsf{isZero}\, t) & := \mathsf{Bool}, & \textit{ha } \mathit{infer}(\Gamma, t) = \mathsf{Nat} \\
 & \mathtt{fail}, & \textit{egyébként} \\
\mathit{infer}(\Gamma, \mathsf{true}) & := \mathsf{Bool} & \\
\mathit{infer}(\Gamma, \mathsf{false}) & := \mathsf{Bool} & \\
\mathit{infer}(\Gamma, \mathsf{if}\, t\, \mathsf{then}\, t'\, \mathsf{else}\, t'') & := \mathit{infer}(\Gamma, t'), & \textit{ha } \mathit{infer}(\Gamma, t) = \mathsf{Bool} \\
 & & \textit{és } \mathit{infer}(\Gamma, t') = \mathit{infer}(\Gamma, t'') \\
 & \mathtt{fail}, & \textit{egyébként} \\
\mathit{infer}(\Gamma, x) & := \mathit{infer}_{\mathsf{Con}}(\Gamma, x) & \\
\mathit{infer}(\Gamma, \mathsf{let}\, t\, \mathsf{in}\, x.t') & := \mathit{infer}((\Gamma, x : A), t'), & \textit{ha } \mathit{infer}(\Gamma, t) = A \in \mathsf{Ty} \\
 & \mathtt{fail}, & \textit{egyébként}
\end{array}
$$

4.4. Feladat. *Bizonyítsuk be a 4.3. lemmát!*

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

4.5. Lemma. *Ha* $\Gamma \vdash t : A$, *akkor* $\Gamma' \vdash t : A$, *feltéve, hogy* Γ' *a* Γ *egy permutációja (ugyanazok a változó-típus párok, csak más sorrendben).*

Bizonyítás. A $\Gamma \vdash t : A$ levezetése szerinti indukcióval.

- A 4.8–4.13 szabályoknál csak az induktív hipotéziseket, majd a szabályt használjuk.
- A 4.15 szabály esetén azt szeretnénk belátni, hogy $\Gamma' \vdash \mathsf{let}\, t_1\, \mathsf{in}\, x.t_2 : A_2$, feltéve, hogy $\Gamma' \vdash t_1 : A_1$ és $\Gamma'' \vdash t_2 : A_2$, ha Γ'' a $\Gamma, x : A_1$ egy permutációja. Mivel Γ' a Γ egy permutációja, ezért a $\Gamma', x : A_1$ a $\Gamma, x : A_1$ permutáci-

ója, így ez az indukciós feltevés azt mondja, hogy $\Gamma', x : A_1 \vdash t_2 : A_2$. Ezt felhasználva a 4.15 szabály alapján megkapjuk, hogy $\Gamma' \vdash \mathsf{let}\, t_1\, \mathsf{in}\, x.t_2 : A_2$.

– A 4.14 szabály esetén azt látjuk be, hogy ha $(x : A) \in \Gamma$, és Γ' a Γ egy permutációja, akkor $(x : A) \in \Gamma'$. Itt egyszerűen annyiszor alkalmazzuk a 4.5 szabályt, ahányadik komponens $(x : A)$ a Γ'-ben (a környezet vége felől számolva).

□

A típusrendszer egy további fontos tulajdonsága a *gyengítési tulajdonság* (weakening).

4.6. Lemma. *Ha $\Gamma \vdash t : A$ levezethető, és $y \notin dom(\Gamma)$, akkor $\Gamma, y : A' \vdash t : A$ is levezethető bármely A'-re.*

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

Bizonyítás. A levezetés szerinti indukcióval. Tehát $P(\Gamma \vdash t : A) = \Gamma, y : A' \vdash \vdash t : A$, ahol $y \notin dom(\Gamma)$. A változó esetében eggyel többször használjuk a 4.7 szabályt, a 4.8,4.11,4.12 szabályok esetén ugyanazeket a szabályokat alkalmazzuk más környezetekre, a 4.9,4.10,4.13 szabályok esetén az indukciós feltevést használjuk, majd magát a szabályt. A 4.15 szabály esetén az indukciós feltevés azt mondja, hogy $\Gamma, y : A' \vdash t_1 : A_1$ és $\Gamma, x : A_1, y : A' \vdash t_2 : A_2$, nekünk pedig azt kell belátnunk, hogy $\Gamma, y : A' \vdash \mathsf{let}\, t_1\, \mathsf{in}\, x.t_2 : A_2$. Mivel $\Gamma, y : A', x : A_1$ a $\Gamma, x : A_1, y : A'$ környezet egy permutációja, a 4.5. lemma alapján megkapjuk, hogy $\Gamma, y : A', x : A_1 \vdash t_2 : A_2$, így alkalmazhatjuk a 4.15 szabályt. □

Helyettesítési lemma (substitution lemma):

4.7. Lemma. *Ha $\Gamma, x : A \vdash t' : A'$ és $\Gamma \vdash t : A$, akkor $\Gamma \vdash t'[x \mapsto t] : A'$.*

Ez a lemma a modularitást fejezi ki: van két külön programunk, $t : A$ és $t' : A'$, utóbbi deklarál egy A típusú változót. t-t A implementációjának, t'-t pedig t kliensének nevezzük. A 4.7. lemma azt mondja, hogy külön-külön típusellenőrizhetjük a két modult, és ekkor összetevés (linking) után is típushelyes lesz a programunk.

Bizonyítás. $\Gamma, x : A \vdash t' : A'$ levezetése szerinti indukció.

– A 4.8,4.11,4.12 szabályok esetén nem csinál semmit a helyettesítés.

– A 4.9,4.10,4.13 szabályok esetén az indukciós feltevésekből következik a helyettesítés helyessége. Például a 4.9 szabály esetén tudjuk, hogy $\Gamma, x : A \vdash t_1, t_2 : \mathsf{Nat}$ és $\Gamma \vdash t_1[x \mapsto t], t_2[x \mapsto t] : \mathsf{Nat}$, és azt szeretnénk belátni, hogy $\Gamma \vdash (t_1 + t_2)[x \mapsto t] : \mathsf{Nat}$. Ez a helyettesítés definíciója alapján megegyezik azzal, hogy $\Gamma \vdash t_1[x \mapsto t] + t_2[x \mapsto t] : \mathsf{Nat}$. Ezt a 4.9 szabály alkalmazásával megkapjuk. A 4.14 szabály alkalmazása esetén a termünk egy változó. Ha ez megegyezik x-szel, akkor a helyettesítés eredménye e lesz, és $A = A'$. Ekkor $\Gamma \vdash e : A$ miatt készen vagyunk. Ha a változónevek nem egyeznek meg, a helyettesítés nem csinál semmit.

– A 4.14 szabály esete triviális.

– A 4.15 szabály esetén az indukciós feltevés alapján tudjuk, hogy $\Gamma \vdash \vdash t_1[x \mapsto t] : A_1$. Ezenkívül $\Gamma, x : A, y : A_1 \vdash t_2 : A_2$, ebből a 4.5. lemma alapján $\Gamma, y : A_1, x : A \vdash t_2 : A_2$, majd az indukciós feltevés alapján kapjuk, hogy $\Gamma, y : A_1 \vdash t_2[x \mapsto t] : A_2$. Mivel $(\mathsf{let}\, t_1 \,\mathsf{in}\, y.t_2)[x \mapsto t] = = \mathsf{let}\, t_1[x \mapsto t] \,\mathsf{in}\, y.t_2[x \mapsto t]$, a 4.15 szabály alapján kapjuk, hogy $\Gamma \vdash \vdash (\mathsf{let}\, t_1 \,\mathsf{in}\, y.t_2)[x \mapsto t] : A_2$.

□

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).

4.8. Lemma. *Ha* $\Gamma \vdash t'[x \mapsto t] : A'$*, akkor minden olyan* A*-ra, melyre* $\Gamma \vdash t : A$*,* $\Gamma, x : A \vdash t' : A'$*.*

Bizonyítás. $\Gamma \vdash t'[x \mapsto t] : A'$ szerinti indukció. □

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

– $(\mathsf{if}\, \mathsf{isZero}\, (\mathsf{num}\, 1)\, \mathsf{then}\, \mathsf{num}\, 2\, \mathsf{else}\, \mathsf{num}\, 3) + \mathsf{let}\, \mathsf{num}\, 1\, \mathsf{in}\, x.x + x$

– $\mathsf{let}\, \mathsf{isZero}\, (\mathsf{num}\, 1)\, \mathsf{in}\, x.x + x$

– $\mathsf{let}\, \mathsf{isZero}\, (\mathsf{num}\, 1)\, \mathsf{in}\, x.\mathsf{if}\, x\, \mathsf{then}\, x\, \mathsf{else}\, \mathsf{false}$

– $\mathsf{isZero}\, (\mathsf{isZero}\, (\mathsf{num}\, 0))$

4.10. Feladat. *Bizonyítsuk be, hogy ha* $\Gamma \vdash t : A$*, akkor* $\Gamma\, \mathsf{wf}$*!*

4.11. Feladat. *Írjuk le a 4.8. lemma bizonyításának részleteit!*

4.2.1. További információ

– A később tanulandó polimorf illetve függő típusrendszerekben nem igaz, hogy a környezet permutálható.

– A lineáris típusrendszerekben nem teljesül a gyengítési tulajdonság.

4.3. Operációs szemantika

A 2.3. fejezetben megadott operációs szemantikát kiegészítjük a let-re vonatkozó szabályokkal. Ezt kétféleképpen tehetjük meg: *érték szerinti* (by value) illetve *név szerinti* (by name) paraméterátadással. Utóbbinál elhagyjuk a zárójelezett 4.16 szabályt és a zárójelezett feltételt a 4.17 szabályból.

$$\left[\frac{t_1 \longmapsto t_1'}{\mathsf{let}\, t_1 \,\mathsf{in}\, x.t_2 \longmapsto \mathsf{let}\, t_1' \,\mathsf{in}\, x.t_2}\right] \tag{4.16}$$

$$\frac{[t_1\ \mathsf{val}]}{\mathsf{let}\, t_1 \,\mathsf{in}\, x.t_2 \longmapsto t_2[x \mapsto t_1]} \tag{4.17}$$

Érték szerinti paraméterátadásnál, mielőtt egy kifejezést hozzákötünk egy változóhoz, azt kiértékeljük. Így legfeljebb 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 kötött változóra, a név szerinti akkor, ha több, mint egyszer hivatkozunk. A kettő előnyeit kombinálja az igény szerinti kiértékelés (call by need, erről nincs szó a jegyzetben).

4.12. Feladat. *Írjunk olyan* let *kifejezést, melynek kiértékelése érték szerinti paraméterátadásnál hatékonyabb (kevesebb átírási lépésből áll).*

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

4.14. Feladat. *Írjunk* let *kifejezést, amelynek kiértékelése ugyanannyi lépésből áll mindkét fajta paraméterátadás esetén.*

A nyílt kifejezések kiértékelése egy adott ponton elakad (a 2.3. fejezetben is voltak elakadó kifejezések, tehát zárt kifejezések kiértékelése is elakadhat).

4.15. Feladat. *Értékeljük ki az* $x + (\mathsf{num}\, 1 + \mathsf{num}\, 2)$ *és az* $(\mathsf{num}\, 1 + \mathsf{num}\, 2) + x$ *kifejezéseket!*

4.4. Típusrendszer és szemantika kapcsolata

A típusmegmaradás és a haladás tételét ebben a típusrendszerben üres környezetre tudjuk kimondani. A bizonyítások ugyanúgy működnek, mint a Nat-Bool nyelv esetén.

4.16. Tétel. *Ha* $\cdot \vdash t : A$ *és* $t \longmapsto t'$*, akkor* $\cdot \vdash t' : A$*.*

4.17. Tétel. *Ha* $\cdot \vdash t : A$*, akkor vagy* $t\,\mathsf{val}$*, vagy létezik olyan* t'*, hogy* $t \longmapsto t'$*.*

5. Függvények

Az előző fejezet típusait kiegészítjük függvény típusokkal.

$$A, A', ... \in \mathsf{Ty} ::= ... \mid A_1 \Rightarrow A_2 \tag{5.1}$$

A – ⇒ – operátor jobbra zárójeleződik, tehát $A_1 \Rightarrow A_2 \Rightarrow A_3 = A_1 \Rightarrow (A_2 \Rightarrow A_3)$.

5.1. Feladat. *Írjunk fel a fenti* Ty *halmaz 10 különböző elemét.*

A termeket kiegészítjük λ absztrakcióval és applikációval hasonló módon, mint ahogyan a 3. fejezetben láttuk.

$$t, t', ... \in \mathsf{Tm} ::= ... \mid \lambda^A x.t \mid t\, t' \tag{5.2}$$

A különbség, hogy a λ operátor a függvény értelmezési tartományának típusát is megkapja paraméterként: aritása $(\mathsf{Ty}, \mathsf{Var}.\mathsf{Tm})\mathsf{Tm}$ (az első paraméterét felső indexbe írjuk), az alkalmazásé $(\mathsf{Tm}, \mathsf{Tm})\mathsf{Tm}$. Emiatt már nem tudjuk megadni a típusrendszert teljesen külön a szintaxistól, hiszen a termekben megjelentek a típusok.

A típusrendszert a következő szabályokkal egészítjük ki.

$$\frac{\Gamma, x : A_1 \vdash t_2 : A_2}{\Gamma \vdash \lambda^{A_1} x.t_2 : A_1 \Rightarrow A_2} \tag{5.3}$$

$$\frac{\Gamma \vdash t : A_1 \Rightarrow A_2 \qquad \Gamma \vdash t_1 : A_1}{\Gamma \vdash t\, t_1 : A_2} \tag{5.4}$$

A 2-vel növelés függvényt például a következőképp adhatjuk meg:

$$\lambda^{\mathsf{Nat}} x.\mathsf{num}\, 2 + x.$$

5.2. Feladat. *Vezesd le, hogy ennek típusa az üres környezetben* $\mathsf{Nat} \Rightarrow \mathsf{Nat}$*!*

5.3. Feladat. *Mutasd meg, hogy nem teljesülne a típusozás unicitása (lásd 2.9.,4.2. lemmák), ha kihagynánk a λ paraméterei közül az értelmezési tartomány típusát.*

Többparaméteres függvényeket ún. *curry-zéssel* tudunk megadni. Például egy $\mathsf{Nat} \Rightarrow (\mathsf{Bool} \Rightarrow \mathsf{Nat})$ típusú függvény bemenete Nat, kimenete pedig egy újabb függvény, mely Bool-ből Nat-ba képez. Úgy is tekinthetünk erre, mint egy két bemenettel, egy Nat és egy Bool típusúval rendelkező függvényre, melynek kimenete Nat. Ezt azzal is kihangsúlyozzuk, hogy a ⇒ operátor jobbra zárójeleződik.

5.4. Feladat. *Adj meg* $\mathsf{Nat} \Rightarrow (\mathsf{Bool} \Rightarrow \mathsf{Nat})$ *típusú függvényt, mely felhasználja mindkét bemenetét.*

5.5. Feladat. *Add meg az* $apply_3 : (\mathsf{Bool} \Rightarrow \mathsf{Bool}) \Rightarrow \mathsf{Bool} \Rightarrow \mathsf{Bool}$ *függvényt, mely az első paramétereként kapott függvényt háromszor alkalmazza a második paraméterében kapott szövegre.*

Az $apply_3$ függvényt *magasabbrendű függvénynek* nevezzük, mert a bemenete függvény. Nem mindegyik programozási nyelv támogat magasabbrendű függvényeket, pedig ez egy fontos absztrakciós eszköz.

Az operációs szemantika a következő szabályokkal egészül ki.

$$\frac{}{\overline{\lambda^A x.t\ \mathsf{val}}} \tag{5.5}$$

$$\frac{t \longmapsto t'}{t\,t_1 \longmapsto t'\,t_1} \tag{5.6}$$

$$\left[\frac{t\ \mathsf{val} \qquad t_1 \longmapsto t_1'}{t\,t_1 \longmapsto t\,t_1'}\right] \tag{5.7}$$

$$\frac{[t_1\ \mathsf{val}]}{(\lambda^A x.t_2)\,t_1 \longmapsto t_2[x \mapsto t_1]} \tag{5.8}$$

A szögletes zárójelezett szabály illetve feltétel az érték szerinti paraméterátadás esetén szükséges. Vegyük észre, hogy ellentétben a 3.2. fejezetben megadottakkal, most nem engedünk tetszőleges átírási sorrendet, az operációs szemantika determinisztikus (vö. 2.21. lemma).

Továbbá igaz a típusmegőrzés (vö. 2.28. és 4.16. tételek) és a haladás (vö. 2.29. és 4.17. tételek) tétele is.

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

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

5.8. Feladat. *Értékeld ki a*

$$\Big(\big(\lambda^{\mathsf{Nat}\Rightarrow\mathsf{Nat}} f.\lambda^{\mathsf{Nat}} x.f\,(f\,x)\big)\,(\lambda^{\mathsf{Nat}} x.x + \mathsf{num}\,1)\Big)\,(\mathsf{num}\,3 + \mathsf{num}\,1)$$

kifejezést érték szerinti paraméterátadással.

Az így megadott jól típusozott termek a 3. fejezetben megadottakkal ellentétben mind terminálnak.

5.9. Lemma. *Ha* $t : A$*, akkor létezik olyan* $t'\ \mathsf{val}$*, hogy* $t \longmapsto^* t'$*.*

6. 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).[1] Lusta szemantika esetén egy tetszőleges $\langle t, t' \rangle$ pár érték, míg mohó szemantika esetén szükséges, hogy t és t' már eleve értékek legyenek.

6.1. Nulláris és bináris szorzat típus

A szintaxis a következő.

$$A, A', ... \in \mathsf{Ty} \quad ::= ... \mid \mathsf{Unit} \mid A \times A' \tag{6.1}$$

$$t, t', ... \quad \in \mathsf{Tm} \; ::= ... \mid \mathsf{tt} \mid \langle t_1, t_2 \rangle \mid \mathsf{proj}_1\, t \mid \mathsf{proj}_2\, t \tag{6.2}$$

A nulláris szorzat típust egyelemű típusnak (top, unit, néha helytelenül void) is nevezik, szokásos jelölései a Unit-on kívül 1 és $()$. Az egyetlen elemét tt-vel (trivially true) jelöljük. A bináris szorzatot (binary product) Descartes-szorzatnak vagy keresztszorzatnak is nevezik.

A $- \times -$ és a $\langle -, - \rangle$ operátorok jobbra zárójeleződnek, tehát például $\mathsf{Unit} \times \times \mathsf{Bool} \times \mathsf{Unit} = \mathsf{Unit} \times (\mathsf{Bool} \times \mathsf{Unit})$.

A típusrendszer a következő.

$$\frac{\Gamma\ \mathsf{wf}}{\Gamma \vdash \mathsf{tt} : \mathsf{Unit}} \tag{6.3}$$

$$\frac{\Gamma \vdash t_1 : A_1 \qquad \Gamma \vdash t_2 : A_2}{\Gamma \vdash \langle t_1, t_2 \rangle : A_1 \times A_2} \tag{6.4}$$

$$\frac{\Gamma \vdash t : A_1 \times A_2}{\Gamma \vdash \mathsf{proj}_1\, t : A_1} \tag{6.5}$$

$$\frac{\Gamma \vdash t : A_1 \times A_2}{\Gamma \vdash \mathsf{proj}_2\, t : A_2} \tag{6.6}$$

Az operációs szemantika a következő.

$$\frac{}{\mathsf{tt}\ \mathsf{val}} \tag{6.7}$$

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

$$\frac{[t_1\ \mathsf{val}] \qquad [t_2\ \mathsf{val}]}{\langle t_1, t_2\rangle\ \mathsf{val}} \tag{6.8}$$

$$\left[\frac{t_1 \longmapsto t_1'}{\langle t_1, t_2\rangle \longmapsto \langle t_1', t_2\rangle}\right] \tag{6.9}$$

$$\left[\frac{t_1\ \mathsf{val} \qquad t_2 \longmapsto t_2'}{\langle t_1, t_2\rangle \longmapsto \langle t_1, t_2'\rangle}\right] \tag{6.10}$$

$$\frac{t \longmapsto t'}{\mathsf{proj}_1\, t \longmapsto \mathsf{proj}_1 t'} \tag{6.11}$$

$$\frac{t \longmapsto t'}{\mathsf{proj}_2\, t \longmapsto \mathsf{proj}_2 t'} \tag{6.12}$$

$$\frac{[t_1\ \mathsf{val}] \qquad [t_2\ \mathsf{val}]}{\mathsf{proj}_1\, \langle t_1, t_2\rangle \longmapsto t_1} \tag{6.13}$$

$$\frac{[t_1\ \mathsf{val}] \qquad [t_2\ \mathsf{val}]}{\mathsf{proj}_2\, \langle t_1, t_2\rangle \longmapsto t_2} \tag{6.14}$$

A szögletes zárójelbe tett feltételek és szabályok csak a mohó szemantikában használatosak. A $-,-$ operátor a bevezető operátor, míg a proj_1 és proj_2 az eliminátorok. Mohó kiértékelés esetén az utasítás szabályok (6.13–6.14) csak akkor működnek, ha a $-,-$ operátor mindkét paramétere ki van értékelve.

6.1. Feladat. *Hány olyan kifejezés van, mely mohó operációs szemantikában érték, és* $(\mathsf{Unit} \times \mathsf{Unit}) \times \mathsf{Unit}$ *típusú illetve* $\mathsf{Unit} \times (\mathsf{Unit} \times \mathsf{Unit})$ *típusú?*

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

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

6.3. Feladat. *Bizonyítsd be a típusmegőrzés és a haladás tételét erre a típusrendszerre.*

A szorzat típusokkal kapunk egy másik módszert a Curry-zés mellett, mellyel meg tudunk adni többparaméteres illetve több visszatérési értékű függvényeket.

6.4. Feladat. *Add meg a három-paraméteres összeadás függvényt, melynek típusa* $(\mathsf{Nat} \times (\mathsf{Nat} \times \mathsf{Nat})) \Rightarrow \mathsf{Nat}$.

6.5. Feladat. *Add meg az ugyanezt végző függvényt a következő típussal* $\mathsf{Nat} \Rightarrow \mathsf{Nat} \Rightarrow \mathsf{Nat} \Rightarrow \mathsf{Nat}$.

6.6. Feladat. *Add meg azt a függvényt, mely visszaadja a bemenetnél eggyel illetve kettővel nagyobb számot. Típusa:* $\mathsf{Nat} \Rightarrow (\mathsf{Nat} \times \mathsf{Nat})$.

6.2. 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.

$$A, A', ... \in \mathsf{Ty} ::= ... \mid \Pi(i_1 \hookrightarrow A_1, ..., i_n \hookrightarrow A_n) \tag{6.15}$$

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 például $(i \hookrightarrow a, j \hookrightarrow b)$ megegyezik $(j \hookrightarrow b, i \hookrightarrow a)$-val.

A kifejezések a következők.

$$t, t', ... \in \mathsf{Tm} ::= ... \mid \langle i_1 \hookrightarrow t_1, ..., i_n \hookrightarrow t_n \rangle \mid \mathsf{proj}_{i_k}\, t \tag{6.16}$$

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

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

$$\Pi\big(\mathsf{gender} \hookrightarrow \mathsf{Bool}, \mathsf{age} \hookrightarrow \mathsf{Nat}, \mathsf{postcode} \hookrightarrow \mathsf{Nat}\big)$$

Egy eleme így adható meg:

$$\langle \mathsf{name} \hookrightarrow \mathsf{false}, \mathsf{age} \hookrightarrow 38, \mathsf{postcode} \hookrightarrow 1092 \rangle$$

A típusrendszer a következő.

$$\frac{\Gamma\ \mathsf{wf} \qquad \Gamma \vdash t_1 : A_1 \qquad ... \qquad \Gamma \vdash t_n : A_n}{\Gamma \vdash \langle i_1 \hookrightarrow t_1, ..., i_n \hookrightarrow t_n \rangle : \Pi(i_1 \hookrightarrow A_1, ..., i_n \hookrightarrow A_n)} \tag{6.17}$$

$$\frac{\Gamma \vdash t : \Pi(i_1 \hookrightarrow A_1, ..., i_n \hookrightarrow A_n)}{\Gamma \vdash \mathsf{proj}_{i_k}\, t : A_k} \tag{6.18}$$

$\Pi(i_1 \hookrightarrow A_1, ..., i_n \hookrightarrow A_n)$-nek ugyanannyi eleme van, mint az $A_1 \times ... \times A_n$ 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:

$$\frac{[t_1\ \mathsf{val} \qquad ... \qquad t_n\ \mathsf{val}]}{\langle i_1 \hookrightarrow t_1, ..., i_n \hookrightarrow t_n \rangle\ \mathsf{val}} \tag{6.19}$$

$$\left[\begin{array}{l} \dfrac{t_1\ \mathsf{val} \qquad ... \qquad t_{k-1}\ \mathsf{val} \qquad t_k \longmapsto t'_k}{\begin{array}{ll} & \langle i_1 \hookrightarrow t_1, ..., i_n \hookrightarrow t_n \rangle \\ \longmapsto & \langle i_1 \hookrightarrow t_1, ..., i_{k-1} \hookrightarrow t_{k-1}, i_k \hookrightarrow t'_k, i_{k+1} \hookrightarrow t_{k+1}, ..., i_n \hookrightarrow t_n \rangle \end{array}} \end{array}\right] \tag{6.20}$$

$$\frac{t \longmapsto t' \qquad k \in \{1, ..., n\}}{\mathsf{proj}_{i_k}\ t \longmapsto \mathsf{proj}_{i_k}\ t'} \tag{6.21}$$

$$\frac{[t_1\ \mathsf{val} \qquad ... \qquad t_n\ \mathsf{val}]}{\mathsf{proj}_{i_k}\ \langle i_1 \hookrightarrow t_1, ..., i_n \hookrightarrow t_n \rangle \longmapsto t_{i_k}} \tag{6.22}$$

A szögletes zárójellel megadott feltételek és szabály csak mohó kiértékelés esetén használható.

6.7. Feladat. *Jelöld be, hogy melyik a bevezető operátor, eliminátor, melyek a sorrendi és melyek az utasítás szabályok.*

6.8. Feladat. *A nulláris és a bináris szorzat típusokat megkapjuk, ha I az üres halmaz illetve a kételemű halmaz. Mutasd meg, hogy az így megadott nulláris és bináris típus ugyanúgy viselkedik típusrendszer és operációs szemantika szempontjából, mint a 6.1. alfejezetben megadott.*

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

7. Összeg típusok

7.1. Nulláris és bináris összeg típus

Szintaxis.

$$A, A', ... \in \mathsf{Ty} \quad ::= ... \mid \mathsf{Empty} \mid A_1 + A_2 \tag{7.1}$$

$$t, t', ... \quad \in \mathsf{Tm} \; ::= ... \mid \mathsf{abort}^A\, t \mid \mathsf{inj}_1{}^{A_1,A_2}\, t \mid \mathsf{inj}_2{}^{A_1,A_2}\, t \mid \mathsf{case}\, t\, x_1.t_1\, x_2.t_2 \tag{7.2}$$

A nulláris összeg típus (bottom, 0 típus, void típus) egy olyan választási lehetőséget ad meg, ahol egy alternatíva sincs. Emiatt nincs bevezető operátora. 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 inj_1 és inj_2. Ezek aritása $(\mathsf{Ty}, \mathsf{Ty}, \mathsf{Tm})\mathsf{Tm}$. Egy összeg típusú kifejezést esetszétválasztással (case) tudunk eliminálni, aritása $(\mathsf{Tm}, \mathsf{Var.Tm}, \mathsf{Var.Tm})\mathsf{Tm}$.

Típusrendszer.

$$\frac{\Gamma \vdash t : \mathsf{Empty}}{\Gamma \vdash \mathsf{abort}^A\, t : A} \tag{7.3}$$

$$\frac{\Gamma \vdash t_1 : A_1}{\Gamma \vdash \mathsf{inj}_1{}^{A_1,A_2}\, t_1 : A_1 + A_2} \tag{7.4}$$

$$\frac{\Gamma \vdash t_2 : A_2}{\Gamma \vdash \mathsf{inj}_2{}^{A_1,A_2}\, t_2 : A_1 + A_2} \tag{7.5}$$

$$\frac{\Gamma \vdash t : A_1 + A_2 \qquad \Gamma, x_1 : A_1 \vdash t_1 : A \qquad \Gamma, x_2 : A_2 \vdash t_2 : A}{\Gamma \vdash \mathsf{case}\, t\, x_1.t_1\, x_2.t_2 : A} \tag{7.6}$$

Operációs szemantika. Az inj_1 és inj_2 típusparamétereit nem írtuk ki a rövidség kedvéért, de ha precízek akarunk lenni, akkor ki kell őket írni.

$$\frac{t \longmapsto t'}{\mathsf{abort}^A\, t \longmapsto \mathsf{abort}^A\, t'} \tag{7.7}$$

$$\frac{[t\,\mathsf{val}]}{\mathsf{inj}_1\, t\,\mathsf{val}} \tag{7.8}$$

$$\frac{[t\,\mathsf{val}]}{\mathsf{inj}_2\, t\,\mathsf{val}} \tag{7.9}$$

$$\left[\frac{t \longmapsto t'}{\mathsf{inj}_1\, t \longmapsto \mathsf{inj}_1\, t'}\right] \tag{7.10}$$

$$\left[\frac{t \longmapsto t'}{\mathsf{inj}_2\, t \longmapsto \mathsf{inj}_2\, t'}\right] \tag{7.11}$$

$$\frac{t \longmapsto t'}{\mathsf{case}\, t\, x_1.t_1\, x_2.t_2 \longmapsto \mathsf{case}\, t'\, x_1.t_1\, x_2.t_2} \tag{7.12}$$

$$\frac{[t\,\mathsf{val}]}{\mathsf{case}\,(\mathsf{inj}_1\, t)\, x_1.t_1\, x_2.t_2 \longmapsto t_1[x_1 \mapsto t]} \tag{7.13}$$

$$\frac{[t\,\mathsf{val}]}{\mathsf{case}\,(\mathsf{inj}_2\, t)\, x_1.t_1\, x_2.t_2 \longmapsto t_2[x_2 \mapsto t]} \tag{7.14}$$

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

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

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

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

7.2. Általános véges összeg típusok

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

$$A, A', ... \in \mathsf{Ty} \;\; ::= ... \,|\, \Sigma(i_1 \hookrightarrow A_1, ..., i_n \hookrightarrow A_n) \tag{7.15}$$

$$t, t', ... \quad \in \mathsf{Tm} \; ::= ... \,|\, \mathsf{inj}_{i_k}{}^{i_1 \hookrightarrow A_1, ..., i_n \hookrightarrow A_n}\, t \,|\, \mathsf{case}\, t\, (i_1 \hookrightarrow x_1.t_1, ..., i_n \hookrightarrow x_n.t_n) \tag{7.16}$$

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

Típusrendszer.

$$\frac{\Gamma \vdash t : A_k}{\Gamma \vdash \mathsf{inj}_{i_k}{}^{i_1 \hookrightarrow A_1, ..., i_n \hookrightarrow A_n}\, t : \Sigma(i_1 \hookrightarrow A_1, ..., i_n \hookrightarrow A_n)} \tag{7.17}$$

$$\frac{\Gamma \vdash t : \Sigma(i_1 \hookrightarrow A_1, ..., i_n \hookrightarrow A_n) \quad \Gamma, x_1 : A_1 \vdash t_1 : A \quad ... \quad \Gamma, x_n : A_n \vdash t_n : A}{\Gamma \vdash \mathsf{case}\, t\, x_1.t_1\, ...\, x_n.t_n : A} \tag{7.18}$$

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

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

7.3. Szorzat és összeg típusok alkalmazásai

Fontos megkülönböztetni a Unit és az Empty típusokat. A Unit-nak pontosan egy eleme van (ezt úgy értjük, hogy egy olyan kifejezés van, mely érték, és

típusa Unit), 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. Az Empty-nak nincs egy eleme sem, ez a teljes információ: ha van egy Empty típusú értékünk (ami lehetetlen), akkor tetszőleges más típusú értéket létre tudunk hozni (abort művelet).

Példaképpen megadunk három típust.

a) A Bool típust megadhatjuk úgy, hogy $\mathsf{Unit} + \mathsf{Unit}$.

b) Felsorolások, például Észak, Dél, Kelet, Nyugat. Ez tkp. $\mathsf{Unit} + \mathsf{Unit} + {+}\mathsf{Unit}{+}\mathsf{Unit}$ típus. Ilyen a beépített karakter típus is, általában egyenlőségvizsgálattal van megadva a case ezekre.

c) Opt típus, mely opcionálisan tartalmaz valamilyen értéket. Szintaxis.

$$A, A', ... \in \mathsf{Ty} \quad ::= ... \mid \mathsf{Opt}\, A \tag{7.19}$$

$$t, t', ... \quad \in \mathsf{Tm} \; ::= ... \mid \mathsf{null}^A \mid \mathsf{just}\, t \mid \mathsf{ifnull}\, t\, t_1\, x.t_2 \tag{7.20}$$

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

$$\frac{\Gamma\ \mathsf{wf}}{\Gamma \vdash \mathsf{null}^A : \mathsf{Opt}\, A} \tag{7.21}$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \mathsf{just}\, t : \mathsf{Opt}\, A} \tag{7.22}$$

$$\frac{\Gamma \vdash t : \mathsf{Opt}\, A \qquad \Gamma \vdash t_1 : A' \qquad \Gamma, x : A \vdash t_2 : A'}{\Gamma \vdash \mathsf{ifnull}\, t\, t_1\, x.t_2 : A'} \tag{7.23}$$

Operációs szemantika.

$$\frac{}{\mathsf{null}^A\ \mathsf{val}} \tag{7.24}$$

$$\frac{[t\ \mathsf{val}]}{\mathsf{just}\, t\ \mathsf{val}} \tag{7.25}$$

$$\left[\frac{t \longmapsto t'}{\mathsf{just}\, t \longmapsto \mathsf{just}\, t'}\right] \tag{7.26}$$

$$\frac{t \longmapsto t'}{\mathsf{ifnull}\, t\, t_1\, x.t_2 \longmapsto \mathsf{ifnull}\, t'\, t_1\, x.t_2} \tag{7.27}$$

$$\frac{}{\mathsf{ifnull}\, \mathsf{null}^A\, t_1\, x.t_2 \longmapsto t_1} \tag{7.28}$$

$$\frac{[t\ \mathsf{val}]}{\mathsf{ifnull}\, (\mathsf{just}\, t)\, t_1\, x.t_2 \longmapsto t_2[x \mapsto t]} \tag{7.29}$$

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

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

A_1 és A_2 típusokat *izomorfnak* nevezzük, ha megadhatók olyan $\cdot \vdash t_{12} : A_1 \Rightarrow A_2$ és $\cdot \vdash t_{21} : A_2 \Rightarrow A_1$ termek, melyekre bármely $\cdot \vdash t_1 : A_1$, $t_1\,\mathsf{val}$ értékre igaz, hogy

$$t_{21}\,(t_{12}\,t_1) \longmapsto^* t_1$$

és bármely $\cdot \vdash t_2 : A_2$, $t_2\,\mathsf{val}$ értékre igaz, hogy

$$t_{12}\,(t_{21}\,t_2) \longmapsto^* t_2.$$

7.9. Feladat. *Döntsd el, hogy az alábbi típusok izomorfak-e, ha igen, add meg t_{12}-őt és t_{21}-et!*

$\mathsf{Bool} \times \mathsf{Unit}$	*és* Bool
$(\mathsf{Unit} + \mathsf{Unit}) + \mathsf{Unit}$	*és* $\mathsf{Unit} + (\mathsf{Unit} + \mathsf{Unit})$
$(A + A') \times A''$	*és* $(A \times A'') + (A' \times A'')$
$A + \mathsf{Empty}$	*és* A
$A \times A'$	*és* $A' \times A$
$A \times \mathsf{Empty}$	*és* Empty

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

Nevezetes a következő típusok izomorfizmusa.

$$A \Rightarrow (B \Rightarrow C) \text{ és } (A \times B) \Rightarrow C$$

A balról jobbra irányt *uncurry*-nek, a jobbról balra irányt *curry*-nek nevezzük.

7.11. Feladat. *Add meg az izomorfizmust!*

8. Konstruktív ítéletlogika

Ebben a fejezetben a konstruktív ítéletlogikával (propozícionális, nulladrendű logikával) és az eddig tanult típusrendszerekkel 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 szabályokkal 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 első, vagy a második komponense igaz. Eliminálni akkor tudunk $A \vee B$-ből C-be, ha A-t illetve 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 (a feltétellista üres).*

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 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
$\top$	Unit
$\bot$	Empty
$A_1 \wedge A_2$	$A_1 \times A_2$
$A_1 \vee A_2$	$A_1 + A_2$
$A_1 \supset A_2$	$A_1 \Rightarrow A_2$
X	lásd 12. fejezet

A levezetési szabályok megfeleltethetők egymásnak: a különbség, hogy ítéletlogikában nem írunk termeket. Például a modus ponens szabály kétféle változata.

$$\frac{\Delta \vdash A \supset B \qquad \Delta \vdash A}{\Delta \vdash B} \qquad \frac{\Gamma \vdash t : A_1 \Rightarrow A_2 \qquad \Gamma \vdash t_1 : A_1}{\Gamma \vdash t\, t_1 : A_2}$$

Ítéletlogikában a bizonyításoknak (melyeknek egy programozási nyelvben a termek felelnek meg) nincs külön szintaxisa, mert 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, például A_1, A_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 (5. fejezet).

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, $|\Delta|^I = \mathsf{t}$ esetén $|A|^I = \mathsf{t}$.

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

8.3. Feladat. *Bizonyítsd be, hogy $\Delta \vdash A$-ból következik $\Delta \models A$*

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 [5], 13. fejezet). Tehát a magasabbrendű függvények és véges típusok operációs szemantikája ehhez 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

Ebben a fejezetben bevezetjük a természetes számok típusát a metaelméleti természetes számok nélkül. A Nat-Bool nyelvben (2. fejezet) a Nat típusnak adhoc módon adtuk meg az operátorait. Most egy általános mechanizmust adunk meg, a *primitív rekurziót*, mellyel sokféle, természetes számokon értelmezett függvényt tudunk megadni. Ezt a típusrendszer Gödeltől származik és System T-nek nevezik.

9.1. Szintaxis

$$\begin{aligned} A, A', ... \in \mathsf{Ty} &::= \mathsf{Nat} \mid A_1 \Rightarrow A_2 \\ t, t', ... \in \mathsf{Tm} &::= x \mid \mathsf{zero} \mid \mathsf{suc}\, t \mid \mathsf{rec}\, t_0\, x.t_1\, t \mid \lambda^A x.t \mid t\, t' \end{aligned}$$

Kétféle természetes szám van (két bevezető operátora van a természetes számoknak): zero (nulla) és $\mathsf{suc}\, e$, amely az e természetes számnak a rákövetkezője. Például 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 $(\mathsf{Tm}, \mathsf{Var}.\mathsf{Tm}, \mathsf{Tm})\mathsf{Tm}$. A rec operátorral tudunk természetes számokon értelmezett függvényt megadni. A $\mathsf{rec}\, t_0\, x.t_1\, t$ kifejezésben a függvényt a t_0 és a t_1 kifejezések határozzák meg, a függvény bemenete pedig t. t_0 azt adja meg, mi legyen a függvény kimenete, ha a bemenet (t) nulla, t_1 pedig azt adja meg, hogy ha a függvény kimenete egy adott számra x, akkor az eggyel nagyobb számra t_1 lesz a függvény kimenete (t_1 persze függhet x-től).

Például az $x \mapsto 2 * x + 1$ függvényt a következőképp adjuk meg. Először is megvizsgáljuk, hogyan működik az első néhány természetes számon.

$$\begin{aligned} 0 &\mapsto 1 \\ 1 &\mapsto 3 \\ 2 &\mapsto 5 \\ 3 &\mapsto 7 \end{aligned}$$

A táblázat alapján azt látjuk, hogy $0 \mapsto \mathsf{suc}\,\mathsf{zero}$, és ha $x \mapsto x_1$, akkor $\mathsf{suc}\, x \mapsto \mathsf{suc}\,(\mathsf{suc}\, x_1)$ (az egymás után következő sorok jobb oldalán levő eredmények közötti különbség 2). A függvényt tehát a következőképp adjuk meg:

$$\lambda^{\mathsf{Nat}} y.\mathsf{rec}\,(\mathsf{suc}\,\mathsf{zero})\,(x_1.\mathsf{suc}\,(\mathsf{suc}\, x_1))\, y$$

9.2. Típusrendszer

A környezetre és változókra vonatkozó szabályok a szokásosak (ahogy a 4.2. fejezetben meg vannak adva).

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

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

$$\frac{\Gamma \vdash t_0 : A \qquad \Gamma, x : A \vdash t_1 : A \qquad \Gamma \vdash t : \mathsf{Nat}}{\Gamma \vdash \mathsf{rec}\, t_0\, x.t_1\, t : A} \tag{9.3}$$

A függvényekre vonatkozó szabályok a szokásosak.

$$\frac{\Gamma, x : A_1 \vdash t : A_2}{\Gamma \vdash \lambda^{A_1} x.t : A_1 \Rightarrow A_2} \tag{9.4}$$

$$\frac{\Gamma \vdash t : A_1 \Rightarrow A_2 \qquad \Gamma \vdash t_1 : A_1}{\Gamma \vdash t\, t_1 : A_2} \tag{9.5}$$

9.3. Operációs szemantika

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

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

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

$$\frac{}{\lambda^A x.t\ \mathsf{val}} \tag{9.8}$$

$$\left[\frac{t \longmapsto t'}{\mathsf{suc}\, t \longmapsto \mathsf{suc}\, t'}\right] \tag{9.9}$$

$$\frac{t \longmapsto t'}{\mathsf{rec}\, t_0\, x.t_1\, t \longmapsto \mathsf{rec}\, t_0\, x.t_1\, t'} \tag{9.10}$$

$$\frac{}{\mathsf{rec}\, t_0\, x.t_1\, \mathsf{zero} \longmapsto t_0} \tag{9.11}$$

$$\frac{\mathsf{suc}\, t\ \mathsf{val}}{\mathsf{rec}\, t_0\, x.t_1\, (\mathsf{suc}\, t) \longmapsto t_1[x \mapsto \mathsf{rec}\, t_0\, x.t_1\, t]} \tag{9.12}$$

$$\frac{t \longmapsto t'}{t\, t_1 \longmapsto t'\, t_1} \tag{9.13}$$

$$\left[\frac{t\ \mathsf{val} \qquad t_1 \longmapsto t_1'}{t\,t_1 \longmapsto t\,t_1'}\right] \tag{9.14}$$

$$\frac{[t_1\ \mathsf{val}]}{(\lambda^A x.t_2)\,t_1 \longmapsto t_2[x \mapsto t_1]} \tag{9.15}$$

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.

Az összeadás függvényt az alábbi módon adjuk meg.

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

Ez a következőképp működik: y-t és z-t akarjuk összeadni, emiatt y-on végzünk rekurziót. Ha $y = \mathsf{zero}$, akkor z-t adunk vissza (hiszen $0 + z$ egyenlő z-vel). Ha $y = \mathsf{suc}\,t$, akkor x-ben megkötjük $\mathsf{rec}\,z\,(x.\mathsf{suc}\,x)\,t$ eredményét, majd hozzáadunk egyet. Például 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 ezzel a szintaxissal is írhatnánk:

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

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). Vigyázz, nehéz!*

9.2. Feladat. *Add meg a természetes számok szorzását! (Egy olyan t kifejezést, mely $\mathsf{Nat} \Rightarrow (\mathsf{Nat} \Rightarrow \mathsf{Nat})$ típusú, és $(t\,\overline{n})\,\overline{n'}$ több lépésben $\overline{n * n'}$-re íródik át.)*

9.3. Feladat. *Bizonyítsd be, hogy bármely $n \in \mathbb{N}$-re $\mathit{plus}\,\overline{n}\,\mathsf{zero} \longmapsto^* \overline{n}$.*

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

9.5. Feladat. *Tegyük fel, hogy a nyelvünkben van Bool típus is (ahogy a 2. fejezetben meg van adva). Írj olyan $\mathsf{Nat} \Rightarrow \mathsf{Nat} \Rightarrow \mathsf{Bool}$ függvényt, mely eldönti, hogy a két paramétere egyenlő-e! Tehát, ha a két természetes szám egyenlő, térjen vissza true-val, egyébként false-szal!*

9.4. Definiálható függvények

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

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

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

System T-ben nem tudunk írni végtelen ciklust (vö. 2.25. lemma).

9.7. Tétel. *Ha $\cdot \vdash t : A$, akkor létezik egy olyan t', hogy $t'\,\mathsf{val}$ és $t \longmapsto^* t'$.*

Emiatt a System T-beli $A_1 \Rightarrow A_2$ típusú kifejezések úgy viselkednek, mint a matematikai függvények.

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

A bizonyítás a (Cantor-féle) átlós metszés technikáját használja (hasonló elven végezhető 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 termet egy számmal kódol. Ennek a részleteibe nem megyünk bele, de a lényeg, hogy minden t termnek megfelel egy $\ulcorner t \urcorner$ természetes szám, és két különböző termet különböző természetes számokkal reprezentálunk.

A következő metaelméleti függvényt adjuk meg. $u \in \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ tetszőleges $t : \mathsf{Nat} \Rightarrow \mathsf{Nat}$ kifejezésre úgy van megadva, hogy $t\,\overline{m} \longmapsto^* \overline{u(\ulcorner t \urcorner, m)}$. Tehát u egy olyan függvény, mely tetszőleges $t : \mathsf{Nat} \Rightarrow \mathsf{Nat}$ 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.

Tegyük fel, hogy u definiálható System T-ben, mégpedig a $t_u : \mathsf{Nat} \Rightarrow \mathsf{Nat} \Rightarrow \mathsf{Nat}$ kifejezéssel, vagyis a következő igaz: $t_u\,\overline{m}\,\overline{n} \longmapsto^* \overline{u(m,n)}$.

Ekkor megadjuk a $t_\Delta : \mathsf{Nat} \Rightarrow \mathsf{Nat}$ kifejezést a következőképp:

$$t_\Delta := \lambda^{\mathsf{Nat}} x.\mathsf{suc}\,(t_u\,x\,x).$$

Egyrészt tudjuk, hogy u-ra teljesül, hogy

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

másrészt t_Δ definíciójából és t_u tulajdonságából

$$t_\Delta\,\overline{\ulcorner t_\Delta \urcorner} \longmapsto \mathsf{suc}\,(t_u\,\overline{\ulcorner t_\Delta \urcorner}\,\overline{\ulcorner t_\Delta \urcorner}) \longmapsto^* \mathsf{suc}\,\big(\overline{u(\ulcorner t_\Delta \urcorner, \ulcorner t_\Delta \urcorner)}\big).$$

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. Generikus programozás

Tegyük fel, hogy van egy $\mathsf{Bool} \Rightarrow \mathsf{Nat}$ függvényünk, például ami true-hoz 1-t, false-hoz 0-t rendel (például $\lambda^{\mathsf{Bool}} x.\mathsf{if}\ x\ \mathsf{then\ num}\ 1\ \mathsf{else\ num}\ 0$). Ezt tudjuk alkalmazni egy Bool típusú kifejezésre. De hasonlóképp tudnánk alkalmazni $\mathsf{Bool} \times \times\ \mathsf{Nat}$ típusú kifejezésre is, úgy, hogy a Nat rész nem változik, tehát $\langle t, t' \rangle$-ből $\langle \mathsf{if}\ t\ \mathsf{then\ num}\ 1\ \mathsf{else\ num}\ 0, t' \rangle$ lesz. Vagy egy $(\mathsf{Nat} + \mathsf{Bool}) \times \mathsf{Bool}$ kifejezésre is, ahol mindkét Bool-ra alkalmaznánk, a Nat típusú értéket meg változatlanul hagynánk. Általánosságban: tetszőleges $A' \Rightarrow A''$ függvényt szeretnénk kiterjeszteni egy α-t tartalmazó A típusra, úgy, hogy egy $A[\alpha \mapsto A'] \Rightarrow A[\alpha \mapsto A'']$ függvényt kapjunk. A generikus programozás ezt teszi lehetővé bizonyos A típusok esetén. Ha A-ban α-n kívül (nulláris és bináris) szorzat és összeg típusok lehetnek, akkor A-t polinomiális típusoperátornak nevezzük. Ha függvény típus is lehet benne, akkor a pozitív típusoperátorok lesznek azok, melyekre a generikus művelet megadható.

10.1. Polinomiális típusoperátorok

A polinomiális típusoperátorok az algebrában tanult polinomokhoz hasonlóak, például

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

helyett azt írjuk, hogy

$$(\mathsf{Unit} + \mathsf{Unit}) \times (\alpha \times \alpha \times \alpha) + \alpha \times \alpha + (\mathsf{Unit} + \mathsf{Unit} + \mathsf{Unit}) \times \alpha + \mathsf{Unit}.$$

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

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

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

$$\frac{\alpha.A_1\ \mathsf{poly} \qquad \alpha.A_2\ \mathsf{poly}}{\alpha.A_1 \times A_2\ \mathsf{poly}} \tag{10.3}$$

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

$$\frac{\alpha.A_1\ \mathsf{poly} \qquad \alpha.A_2\ \mathsf{poly}}{\alpha.A_1 + A_2\ \mathsf{poly}} \tag{10.5}$$

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

10.2. Feladat. *Mi lesz* $\Big(\mathsf{Empty} + \big((\mathsf{Unit} \times \alpha) + \alpha \times \alpha\big)\Big)[\alpha \mapsto (\mathsf{Nat} + \mathsf{Bool})]$ *eredménye?*

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

Az $A' \Rightarrow A''$ függvény kiterjesztését a map operátorral adjuk meg. Függvények helyett az $x : A' \vdash t'' : A''$ kifejezést használunk.

$$t, t', ... \in \mathsf{Tm} ::= ... \mid \mathsf{map}^{\alpha.A}(x'.t'')\, t \tag{10.6}$$

$$\frac{\alpha.A\ \mathsf{poly} \qquad \Gamma, x' : A' \vdash t'' : A'' \qquad \Gamma \vdash t : A[\alpha \mapsto A']}{\Gamma \vdash \mathsf{map}^{\alpha.A}(x'.t'')\, t : A[\alpha \mapsto A'']} \tag{10.7}$$

A t kifejezésben vannak A' típusú részkifejezések, ezeket fogja a map operátor A'' típusú részkifejezésekre cserélni. A t'' művelet (mely A'-ből A''-be képez), mondja meg, hogy hogyan kell őket lecserélni. Az operációs szemantika azt fejezi ki, hogy ott kell végrehajtanunk a t'' műveletet, ahol α volt a polinomban.

$$\overline{\mathsf{map}^{\alpha.\alpha}(x'.t'')\, t \longmapsto t''[x' \mapsto t]} \tag{10.8}$$

$$\overline{\mathsf{map}^{\alpha.\mathsf{Unit}}(x'.t'')\, t \longmapsto t} \tag{10.9}$$

$$\overline{\mathsf{map}^{\alpha.A_1 \times A_2}(x'.t'')\, t \longmapsto \langle \mathsf{map}^{\alpha.A_1}(x'.t'')\, (\mathsf{proj}_1\, t), \mathsf{map}^{\alpha.A_2}(x'.t'')\, (\mathsf{proj}_2\, t)\rangle} \tag{10.10}$$

$$\overline{\mathsf{map}^{\alpha.\mathsf{Empty}}(x'.t'')\, t \longmapsto t} \tag{10.11}$$

$$\begin{array}{ll} & \mathsf{map}^{\alpha.A_1+A_2}(x'.t'')\, t \\ \longmapsto & \mathsf{case}\, t\, x_1.\big(\mathsf{inj}_1\, (\mathsf{map}^{\alpha.A_1}(x'.t'')\, x_1)\big)\, x_2.\big(\mathsf{inj}_2\, (\mathsf{map}^{\alpha.A_2}(x'.t'')\, x_2)\big) \end{array} \tag{10.12}$$

10.3. Feladat. *Mutasd meg, hogy ha az összeg és szorzat típusok kiértékelése szigorú, akkor bármely t termre*

$$\mathsf{map}^{\alpha.\mathsf{Unit}+(\mathsf{Bool}\times\alpha)}(x'.\mathsf{suc}\, x')\, \big(\mathsf{inj}_2\, \langle \mathsf{true}, t\rangle\big) \longmapsto^* \mathsf{inj}_2\, \langle \mathsf{true}, \mathsf{suc}\, t\rangle !$$

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

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

10.6. Feladat. *Mutasd meg, hogy a konstans típusoperátorral végzett* map *nem csinál semmit. Vagyis, ha $t\ \mathsf{val}$ és $t : A$, akkor $\mathsf{map}^{\alpha.A}(x'.t'')\, t \longmapsto^* t$ függetlenül attól, hogy mi t''.*

10.2. Pozitív típusoperátorok

Szeretnénk függvénytípusokat is megengedni a típusoperátorokban. Ezeket megszorítjuk a szigorúan pozitív (strictly positive) típusoperátorokra: egy $\alpha.A_1 \Rightarrow A_2$ típusoperátor szigorúan pozitív, ha A_1-ben nem szerepel α és $\alpha.A_2$ is szigorúan pozitív. A pozitív, de nem szigorú típusoperátorokkal nem foglalkozunk.

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

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

$$\frac{}{\alpha.\mathsf{Unit}\,\mathsf{spos}} \tag{10.14}$$

$$\frac{\alpha.A_1\,\mathsf{spos} \qquad \alpha.A_2\,\mathsf{spos}}{\alpha.A_1 \times A_2\,\mathsf{spos}} \tag{10.15}$$

$$\frac{}{\alpha.\mathsf{Empty}\,\mathsf{spos}} \tag{10.16}$$

$$\frac{\alpha.A_1\,\mathsf{poly} \qquad \alpha.A_2\,\mathsf{spos}}{\alpha.A_1 + A_2\,\mathsf{spos}} \tag{10.17}$$

$$\frac{A_1\,\mathsf{type} \qquad \alpha.A_2\,\mathsf{spos}}{\alpha.A_1 \Rightarrow A_2\,\mathsf{spos}} \tag{10.18}$$

A polinomiális típusoperátorokhoz képest csak az utolsó szabály új. $A_1\,\mathsf{type}$ azt jelenti, hogy A_1 egy típus és nem szerepel benne típusváltozó. Ennek levezetési szabályai:

$$\frac{}{\mathsf{Unit}\,\mathsf{type}} \tag{10.19}$$

$$\frac{}{\mathsf{Empty}\,\mathsf{type}} \tag{10.20}$$

$$\frac{A_1\,\mathsf{type} \qquad A_2\,\mathsf{type}}{A_1 \times A_2\,\mathsf{type}} \tag{10.21}$$

$$\frac{A_1\,\mathsf{type} \qquad A_2\,\mathsf{type}}{A_1 + A_2\,\mathsf{type}} \tag{10.22}$$

$$\frac{A_1\,\mathsf{type} \qquad A_2\,\mathsf{type}}{A_1 \Rightarrow A_2\,\mathsf{type}} \tag{10.23}$$

A map operátor típusozási szabálya.

$$\frac{\alpha.A\,\mathsf{spos} \qquad \Gamma, x' : A' \vdash t'' : A'' \qquad \Gamma \vdash t : A[\alpha \mapsto A']}{\Gamma \vdash \mathsf{map}^{\alpha.A}(x'.e'')\,t : A[\alpha \mapsto A'']} \tag{10.24}$$

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

$$\frac{}{\mathsf{map}^{\alpha.A_1 \Rightarrow A_2}(x'.t'')\,t \longmapsto \lambda^{A_1} x_1.\mathsf{map}^{\alpha.A_2}(x'.t'')\,(t\,x_1)} \tag{10.25}$$

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

10.2.1. További információ

– A típusoperátorok funktorok, lásd [7].

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

11. 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, melyekre véges sokszor lehet destruktort alkalmazni. 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 korekurzornak).

Ebben a fejezetben először példákat adunk induktív és koinduktív típusokra (egy példát már láttunk: a természetes számokat a 9. fejezetben), majd megadjuk őket általánosságban.

11.1. Példák induktív és koinduktív típusokra

a) Természetes számok induktív típusát lásd a 9. fejezetben.

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

$$\frac{\Gamma \vdash t : \mathsf{Nat}}{\Gamma \vdash \mathsf{leaf}\, t : \mathsf{Tree}} \tag{11.1}$$

$$\frac{\Gamma \vdash t_1 : \mathsf{Tree} \qquad \Gamma \vdash t_2 : \mathsf{Tree}}{\Gamma \vdash \mathsf{node}\, t_1\, t_2 : \mathsf{Tree}} \tag{11.2}$$

Rekurzor (eliminációs szabály):

$$\frac{\Gamma, x : \mathsf{Nat} \vdash t_1 : A \qquad \Gamma, x_1 : A, x_2 : A \vdash t_2 : A \qquad \Gamma \vdash t : \mathsf{Tree}}{\Gamma \vdash \mathsf{rec}'_{\mathsf{Tree}}\, x.t_1\, x_1.x_2.t_2\, t : A} \tag{11.3}$$

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.

$$\frac{}{\mathsf{leaf}\, t\, \mathsf{val}} \tag{11.4}$$

$$\frac{}{\mathsf{node}\, t_1\, t_2\, \mathsf{val}} \tag{11.5}$$

$$\frac{t \longmapsto t'}{\mathsf{rec}'_{\mathsf{Tree}}\, x.t_1\, x_1.x_2.t_2\, t \longmapsto \mathsf{rec}'_{\mathsf{Tree}}\, x.t_1\, x_1.x_2.t_2\, t'} \tag{11.6}$$

$$\frac{}{\mathsf{rec}'_{\mathsf{Tree}}\, x.t_1\, x_1.x_2.t_2\, (\mathsf{leaf}\, t) \longmapsto t_1[x \mapsto t]} \tag{11.7}$$

$$\begin{array}{ll} & \mathsf{rec}'_{\mathsf{Tree}}\, x.t_1\, x_1.x_2.t_2\, (\mathsf{node}\, t\, t') \\ \longmapsto & t_2[x_1 \mapsto \mathsf{rec}'_{\mathsf{Tree}}\, x.t_1\, x_1.x_2.t_2\, t, x_2 \mapsto \mathsf{rec}'_{\mathsf{Tree}}\, x.t_1\, x_1.x_2.t_2\, t'] \end{array} \tag{11.8}$$

Például a leveleket megszámoló függvény:

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

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

11.2. Feladat. *Írj olyan* $\mathsf{Tree} \Rightarrow \mathsf{Bool}$ *függvényt, mely eldönti, hogy a fa egyik levele tartalmazza-e a* 3 *számot!*

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

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

11.5. Feladat. *Add meg a* map *függvényt listákon, mely egy* $\mathsf{Nat} \Rightarrow \mathsf{Nat}$ *függvényt alkalmaz pontonként a listára!*

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

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

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

11.9. Feladat. *Készíts olyan függvényt, mely egy listában soronként (szélességi bejárással) adja vissza a fa leveleit. (Nehéz.)*

c) 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).

$$\frac{\Gamma \vdash t : \mathsf{Stream}}{\Gamma \vdash \mathsf{head}\, t : \mathsf{Nat}} \tag{11.9}$$

$$\frac{\Gamma \vdash t : \mathsf{Stream}}{\Gamma \vdash \mathsf{tail}\, t : \mathsf{Stream}} \tag{11.10}$$

Egy folyamra gondolhatunk úgy, mint egy állapotautomatára. Az aktuális állapota A típusú. t_1 adja meg, hogy egy $x : A$ állapotból hogyan kapjuk

meg a Nat típusú kimenetet, t_2 pedig azt, hogy az aktuális állapotból hogyan kapjuk meg a következő állapotot. A kezdőállapotot t adja meg.

$$\frac{\Gamma, x : A \vdash t_1 : \mathsf{Nat} \qquad \Gamma, x : A \vdash t_2 : A \qquad \Gamma \vdash t : A}{\Gamma \vdash \mathsf{strgen}\, x.t_1\, x.t_2\, t : \mathsf{Stream}} \tag{11.11}$$

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

$$\frac{}{\mathsf{strgen}\, x.t_1\, x.t_2\, t\, \mathsf{val}} \tag{11.12}$$

A sorrendi szabályok a destruktorok paraméterét értékelik ki.

$$\frac{t \longmapsto t'}{\mathsf{head}\, t \longmapsto \mathsf{head}\, t'} \tag{11.13}$$

$$\frac{t \longmapsto t'}{\mathsf{tail}\, t \longmapsto \mathsf{tail}\, t'} \tag{11.14}$$

Az utasítás szabályok azt adják meg, hogy mi történik, ha egy destruktort alkalmazunk a generátorra. Ha a folyam fejét akarjuk megkapni, akkor a t_1-nek adjuk meg az aktuális állapotát.

$$\frac{}{\mathsf{head}\, (\mathsf{strgen}\, x.t_1\, x.t_2\, t) \longmapsto t_1[x \mapsto t]} \tag{11.15}$$

Ha a folyam farkát akarjuk megkapni, generálunk egy új folyamot, melynek aktuális állapotát t_2-vel léptetjük.

$$\frac{}{\mathsf{tail}\, (\mathsf{strgen}\, x.t_1\, x.t_2\, t) \longmapsto \mathsf{strgen}\, x.t_1\, x.t_2\, (t_2[x \mapsto t])} \tag{11.16}$$

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.

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

11.10. Feladat. *Add meg a* map *függvényt folyamokon, mely egy* Nat ⇒ Nat *függvényt alkalmaz pontonként!*

11.11. Feladat. *Meg lehet-e adni egy olyan* Stream ⇒ Stream *függvényt, mely tetszőleges folyamnak kiszűri az 5-nél kisebb elemeit?*

11.2. A 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 con és egy rec operátorral adjuk meg.

a) Természetes számok.

$$\frac{\Gamma \vdash t : \mathsf{Unit} + \mathsf{Nat}}{\Gamma \vdash \mathsf{con}_{\mathsf{Nat}}\,t : \mathsf{Nat}} \tag{11.17}$$

$$\frac{\Gamma, x : \mathsf{Unit} + A \vdash t_1 : A \qquad \Gamma \vdash e : \mathsf{Nat}}{\Gamma \vdash \mathsf{rec}_{\mathsf{Nat}}\,x.t_1\,t : A} \tag{11.18}$$

$$\frac{}{\mathsf{con}_{\mathsf{Nat}}\,t\;\mathsf{val}} \tag{11.19}$$

$$\frac{t \longmapsto t'}{\mathsf{rec}_{\mathsf{Nat}}\,x.t_1\,t \longmapsto \mathsf{rec}_{\mathsf{Nat}}\,x.t_1\,t'} \tag{11.20}$$

$$\frac{}{\mathsf{rec}_{\mathsf{Nat}}\,x.t_1\,(\mathsf{con}_{\mathsf{Nat}}\,t) \longmapsto t_1[x \mapsto \mathsf{map}^{\alpha.\mathsf{Unit}+\alpha}(y.\mathsf{rec}_{\mathsf{Nat}}\,(x.t_1)\,y)\,t]} \tag{11.21}$$

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

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

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

A rekurzorban (11.18) a t_1 paraméter egyszerre adja meg, hogy mit kell nulla és rákövetkező esetben tenni. Nulla esetén nem kap semmi információt (Unit), rákövetkező esetén megkapja a rekurzív hívás A típusú eredményét. A $\mathsf{con}_{\mathsf{Nat}}$ konstruktor alkalmazása értéket ad meg (11.19), és a sorrendi szabály (11.20) a rekurzor természetes szám paraméterét értékeli ki. Ha a rekurzort a konstruktorra alkalmazzuk (11.21), akkor t_1-re írjuk át az eredményt, ami függ egy $x : \mathsf{Unit} + A$ változótól. Ennek értékét úgy adjuk meg, hogy a generikus map segítségével a A részt helyettesítjük a rekurzív hívással. Ha ezt átírjuk a generikus programozás 10.12., 10.9 és 10.8 szabályait alkalmazva, a következőt kapjuk.

$$\mathsf{rec}_{\mathsf{Nat}}\,x.t_1\,(\mathsf{con}_{\mathsf{Nat}}\,t) \longmapsto t_1[x \mapsto \mathsf{case}\,t\,x_1.(\mathsf{inj}_1\,x_1)\,x_2.(\mathsf{inj}_2\,(\mathsf{rec}_{\mathsf{Nat}}\,(x.t_1)\,x_2))]$$

Tehát case-szel megnézzük, hogy a konstruktor t paramétere nullát vagy rákövetkezőt jelent-e. Az előbbi esetben x-et $\mathsf{inj}_1\,\mathsf{tt}$-vel helyettesítjük ($\mathsf{Unit}$ 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:

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

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.

b) Bináris fák (megjegyezzük, hogy $A_1 + A_2 \times A_3$ zárójelezése $A_1 + (A_2 \times A_3)$).

$$\frac{\Gamma \vdash t : \mathsf{Nat} + \mathsf{Tree} \times \mathsf{Tree}}{\Gamma \vdash \mathsf{con}_{\mathsf{Tree}}\,t : \mathsf{Tree}} \tag{11.22}$$

$$\frac{\Gamma, x : \mathsf{Nat} + A \times A \vdash t_1 : A \qquad \Gamma \vdash t : \mathsf{Tree}}{\Gamma \vdash \mathsf{rec}_{\mathsf{Tree}}\,x.t_1\,t : A} \tag{11.23}$$

$$\frac{}{\mathsf{con}_{\mathsf{Tree}}\,t\,\mathsf{val}} \tag{11.24}$$

$$\frac{t \longmapsto t'}{\mathsf{rec}_{\mathsf{Tree}}\, x.t_1\, t \longmapsto \mathsf{rec}_{\mathsf{Tree}}\, x.t_1\, t'} \tag{11.25}$$

$$\frac{}{\mathsf{rec}_{\mathsf{Tree}}\, x.t_1\, (\mathsf{con}_{\mathsf{Tree}}\, t) \longmapsto t_1[x \mapsto \mathsf{map}^{\alpha.\mathsf{Nat}+\alpha\times\alpha}(y.\mathsf{rec}_{\mathsf{Tree}}\, (x.t_1)\, y)\, t]} \tag{11.26}$$

c) Természetes számok folyamai. A destruktorokat a des 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).

$$\frac{\Gamma \vdash t : \mathsf{Stream}}{\Gamma \vdash \mathsf{des}_{\mathsf{Stream}}\, t : \mathsf{Nat} \times \mathsf{Stream}} \tag{11.27}$$

A folyam generálásához kell egy t_1, ami egy adott $x : A$ állapotra megadja a kimenetet (ami egy természetes szám) és az új állapotot (A). Ezenkívül szükségünk van az aktuális állapotra, ezt t adja meg.

$$\frac{\Gamma, x : A \vdash t_1 : \mathsf{Nat} \times A \qquad \Gamma \vdash t : A}{\Gamma \vdash \mathsf{gen}_{\mathsf{Stream}}\, x.t_1\, t : \mathsf{Stream}} \tag{11.28}$$

Egy generált folyam érték:

$$\frac{}{\mathsf{gen}_{\mathsf{Stream}}\, x.t_1\, t\, \mathsf{val}} \tag{11.29}$$

Az alábbi sorrendi szabály azt mondja, hogy a destruktor paraméterét kiértékelhetjük.

$$\frac{t \longmapsto t'}{\mathsf{des}_{\mathsf{Stream}}\, t \longmapsto \mathsf{des}_{\mathsf{Stream}}\, t'} \tag{11.30}$$

Ha a destruktort a generátorra alkalmazzuk, akkor t_1-nek megadjuk az aktuális állapotot (t-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 $\mathsf{gen}_{\mathsf{Stream}}\, x.t_1$ generátort:

$$\frac{}{\mathsf{des}_{\mathsf{Stream}}\, (\mathsf{gen}_{\mathsf{Stream}}\, x.t_1\, t) \longmapsto \mathsf{map}^{\alpha.\mathsf{Nat}\times\alpha}(y.\mathsf{gen}_{\mathsf{Stream}}\, x.t_1\, y)\, (t_1[x \mapsto t])} \tag{11.31}$$

11.3. Általános induktív és koinduktív típusok

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

A típusokat kiegészítjük típusváltozókkal (α, α', β, fajtájuk Tyvar) és induktív illetve koinduktív típusokkal. ind és coind aritása (Tyvar.Ty)Ty. Tehát egy

konkrét induktív vagy koinduktív típus megadásához meg kell adni egy $\alpha.A$ típusoperátort. Természetes számok esetén ez például $\alpha.\mathsf{Unit} + \alpha$ lesz, bináris fák esetén $\beta.\mathsf{ind}^{\alpha.\mathsf{Unit}+\alpha} + \beta \times \beta$, folyamok esetén $\beta.\mathsf{ind}^{\alpha.\mathsf{Unit}+\alpha} \times \beta$.

$$A, A_1, ... \in \mathsf{Ty} ::= ... \mid \alpha \mid \mathsf{ind}^{\alpha.A} \mid \mathsf{coind}^{\alpha.A} \tag{11.32}$$

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

$$\begin{aligned} t, t', ... \in \mathsf{Tm} ::= ... &\mid \mathsf{con}^{\alpha.A}\, t && \text{konstruktor} \\ &\mid \mathsf{rec}^{\alpha.A}\, t && \text{rekurzor} \\ &\mid \mathsf{des}^{\alpha.A}\, t && \text{destruktor} \\ &\mid \mathsf{gen}^{\alpha.A}\, t && \text{generátor} \end{aligned} \tag{11.33}$$

Mielőtt megadjuk a típusrendszert, az alábbi összefoglaló táblázattal megmutatjuk az induktív és koinduktív típusok közötti dualitást.

$$A[\alpha \mapsto \mathsf{ind}^{\alpha.A}] \xrightarrow{\mathsf{con}} \mathsf{ind}^{\alpha.A} \qquad \mathsf{ind}^{\alpha.A} \xrightarrow{\mathsf{rec}} A'$$

$$\mathsf{coind}^{\alpha.A} \xrightarrow{\mathsf{des}} A[\alpha \mapsto \mathsf{coind}^{\alpha.A}] \qquad A' \xrightarrow{\mathsf{gen}} \mathsf{ind}^{\alpha.A}$$

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

$$\frac{\alpha.A\,\mathsf{spos} \qquad \Gamma \vdash t : A[\alpha \mapsto \mathsf{ind}^{\alpha.A}]}{\Gamma \vdash \mathsf{con}^{\alpha.A}\, t : \mathsf{ind}^{\alpha.A}} \tag{11.34}$$

A rekurzornak meg kell adni egy t induktív típusbeli elemet, és azt, hogy hogyan működjön a különböző konstrukciós formák esetén, melyeket A ad meg. Most A-ban az α típusváltozót A'-vel helyettesítjük, ebbe a típusba eliminálunk, és emiatt a rekurzív hívások eredményei is ilyen típusúak.

$$\frac{\alpha.A\,\mathsf{spos} \qquad \Gamma, x : A[\alpha \mapsto A'] \vdash t_1 : A' \qquad \Gamma \vdash t : \mathsf{ind}^{\alpha.A}}{\Gamma \vdash \mathsf{rec}^{\alpha.A}\, x.t_1\, t : A'} \tag{11.35}$$

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

$$\frac{\alpha.A\,\mathsf{spos} \qquad \Gamma \vdash t : \mathsf{coind}^{\alpha.A}}{\Gamma \vdash \mathsf{des}^{\alpha.A}\, t : A[\alpha \mapsto \mathsf{coind}^{\alpha.A}]} \tag{11.36}$$

Egy koinduktív típusú elemet úgy generálunk, hogy megadjuk az aktuális állapotát ($t : A'$), és hogy mit adnak a destruktorok attól függően, hogy mi az aktuális állapot.

$$\frac{\alpha.A\,\mathsf{spos} \qquad \Gamma, x : A' \vdash t_1 : A[\alpha \mapsto A'] \qquad \Gamma \vdash t : A'}{\Gamma \vdash \mathsf{gen}^{\alpha.A}\, x.t_1\, t : \mathsf{coind}^{\alpha.A}} \tag{11.37}$$

Figyeljük meg az induktív és koinduktív típusokra vonatkozó szabályok szimmetriáját.

Lusta operációs szemantika. Az egyetlen induktív érték a konstuktor.

$$\frac{}{\mathsf{con}^{\alpha.A}\, t\, \mathsf{val}} \tag{11.38}$$

A rekurzor induktív típusú paraméterét kiértékeljük.

$$\frac{t \longmapsto t'}{\mathsf{rec}^{\alpha.A}\, x.t_1\, t \longmapsto \mathsf{rec}^{\alpha.A}\, x.t_1\, t'} \tag{11.39}$$

Ha a rekurzort egy konstruktorra alkalmazzuk, akkor a rekurzor t_1 paraméterét használjuk, ami megmondja, hogy mely konstrukciós forma esetén mit kell csinálni, és ez függ x-től, amibe t-t, a konstrukciós formát magát szeretnénk helyettesíteni. t-ben viszont a rekurzív előfordulásokat (ahol A-ban α van) helyettesíteni kell a rekurzív hívással.

$$\frac{}{\mathsf{rec}^{\alpha.A}\, x.t_1\, (\mathsf{con}^{\alpha.A}\, t) \longmapsto t_1[x \mapsto \mathsf{map}^{\alpha.A}\, y.(\mathsf{rec}^{\alpha.A}\, x.t_1\, y)\, t]} \tag{11.40}$$

Az egyetlen koinduktív érték a generátor.

$$\frac{}{\mathsf{gen}^{\alpha.A}\, x.t_1\, t\, \mathsf{val}} \tag{11.41}$$

A destruktor paraméterét kiértékeljük.

$$\frac{t \longmapsto t'}{\mathsf{des}^{\alpha.A}\, t \longmapsto \mathsf{des}^{\alpha.A}\, t} \tag{11.42}$$

Ha a destruktort a generátorra alkalmazzuk, akkor t_1-et adjuk vissza, mely megadja az eredményt minden destrukciós formára. Viszont ez függ az x-től, melyet az aktuális állapot, $t : A'$ ad meg. A rekurzív előfordulások viszont koinduktív típusúak kell, hogy legyenek, nem A' típusúak, ezért a generikus map segítségével az α helyeken újabb koinduktív típusokat generálunk az új A' állapotokból.

$$\frac{}{\mathsf{des}^{\alpha.A}\, (\mathsf{gen}^{\alpha.A}\, x.t_1\, t) \longmapsto \mathsf{map}^{\alpha.A}\, (y.\mathsf{gen}^{\alpha.A}\, x.t_1\, y)(t_1[x \mapsto t])} \tag{11.43}$$

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

11.3.1. További információ

– Az $\alpha.A\,\mathsf{spos}$ ítéletet a következő levezetési szabályokkal adjuk meg (melyek egyszerre vannak megadva az $A\,\mathsf{type}$ levezetési szabályaival).

$$\frac{}{\alpha.\mathsf{Unit}\,\mathsf{spos}} \tag{11.44}$$

$$\frac{\alpha.A_1\,\mathsf{spos} \qquad \alpha.A_2\,\mathsf{spos}}{\alpha.A_1 \times A_2\,\mathsf{spos}} \tag{11.45}$$

$$\frac{}{\alpha.\mathsf{Empty}\,\mathsf{spos}} \tag{11.46}$$

$$\frac{\alpha.A_1\,\mathsf{spos} \qquad \alpha.A_2\,\mathsf{spos}}{\alpha.A_1 + A_2\,\mathsf{spos}} \tag{11.47}$$

$$\frac{A_1\,\mathsf{type} \qquad \alpha.A_2\,\mathsf{spos}}{\alpha.A_1 \Rightarrow A_2\,\mathsf{spos}} \tag{11.48}$$

$$\frac{\mathsf{ind}^{\beta.A}\,\mathsf{type}}{\alpha.\mathsf{ind}^{\beta.A}\,\mathsf{spos}} \tag{11.49}$$

$$\frac{\mathsf{coind}^{\beta.A}\,\mathsf{type}}{\alpha.\mathsf{coind}^{\beta.A}\,\mathsf{spos}} \tag{11.50}$$

$$\frac{}{\mathsf{Unit}\,\mathsf{type}} \tag{11.51}$$

$$\frac{A_1\,\mathsf{type} \qquad A_2\,\mathsf{type}}{A_1 \times A_2\,\mathsf{type}} \tag{11.52}$$

$$\frac{}{\mathsf{Empty}\,\mathsf{type}} \tag{11.53}$$

$$\frac{A_1\,\mathsf{type} \qquad A_2\,\mathsf{type}}{A_1 + A_2\,\mathsf{type}} \tag{11.54}$$

$$\frac{A_1\,\mathsf{type} \qquad A_2\,\mathsf{type}}{A_1 \Rightarrow A_2\,\mathsf{type}} \tag{11.55}$$

$$\frac{\alpha.A\,\mathsf{spos}}{\mathsf{ind}^{\alpha.A}\,\mathsf{type}} \tag{11.56}$$

$$\frac{\alpha.A\,\mathsf{spos}}{\mathsf{coind}^{\alpha.A}\,\mathsf{type}} \tag{11.57}$$

– Ha $\alpha.A$-t nem szorítjuk meg szigorúan pozitív típuskifejezésekre, hanem negatív típuskifejezéseket is megengedünk, akkor az üres környezetben

meg tudunk adni Empty típusú elemet:

$$
\begin{array}{l}
\cdot \vdash \mathsf{let}\ \lambda^{\mathsf{Empty}} x.x\ \mathsf{in}\ f. \\
\quad \mathsf{let}\ \lambda^{\mathsf{ind}^{\alpha.\alpha\Rightarrow\mathsf{Empty}}} y.\mathsf{rec}^{\alpha.\alpha\Rightarrow\mathsf{Empty}}\ x.x\ y\ \mathsf{in}\ app. \\
\quad\quad \mathsf{let}\ \mathsf{con}^{\alpha.\alpha\Rightarrow\mathsf{Empty}}\ (\lambda^{\mathsf{ind}^{\alpha.\alpha\Rightarrow\mathsf{Empty}}} x.f\ (app\ x\ x))\ \mathsf{in}\ h. \\
\quad\quad\quad app\ h\ h \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad : \mathsf{Empty}
\end{array}
$$

Ezt az elemet viszont nem tudjuk kiértékelni, mert az üres típusnak nincs egy értéke sem.

11.4. További példák

A környezetfüggetlen nyelvtannal megadott formális nyelvek induktív típusok.

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

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

A következő rövidítésekkel adható meg induktív típusokkal és konstruktorokkal.

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

Néhány példa induktív típusokra:

$\mathsf{ind}^{\alpha.\mathsf{Unit}+\alpha}$	természetes számok
$\mathsf{ind}^{\alpha.\mathsf{Unit}+A\times\alpha}$	A típusú elemeket tartalmazó listák
$\mathsf{ind}^{\alpha.\mathsf{Unit}+\alpha\times\alpha}$	bináris fák, a leveleknél nincs információ
$\mathsf{ind}^{\alpha.\mathsf{Unit}+(\mathsf{Nat}\Rightarrow\alpha)}$	minden lépésben végtelen-felé ágazó fák
$\mathsf{ind}^{\alpha.\mathsf{Unit}+(\alpha\times\alpha)+(\alpha\times\alpha\times\alpha)}$	kétfelé és háromfelé is elágazó fák
$\mathsf{ind}^{\alpha.\mathsf{Nat}+\alpha\times\alpha+\alpha+\mathsf{Unit}+\mathsf{Unit}+\alpha\times\alpha\times\alpha}$	számok és logikai értékek kifejezései
$\mathsf{ind}^{\alpha.\mathsf{Nat}+\mathsf{Nat}\times\alpha+\alpha\times\alpha}$	lambda-kalkulus De Bruijn indexekkel

11.13. Feladat. *A kotermészetes számok típusa* $\mathsf{Conat} = \mathsf{coind}^{\alpha.\mathsf{Unit}+\alpha}$. *Ezek potenciálisan végtelen nagy számok. Tudunk-e* $\mathsf{Conat} \Rightarrow \mathsf{Opt}\, Nat$ *függvényt írni, mely megadja, mekkora volt a kotermészetes szám (végtelen esetén* null*-t adna vissza).*

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

11.4.1. 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 például:

$$A \Rightarrow bA \,|\, a \,|\, cS$$
$$S \Rightarrow aSA \,|\, A$$

12. Polimorfizmus

A típusokat arra használtuk, hogy kiszűrjük az értelmetlen programokat. Ennek viszont a kód bőbeszédűségével (verbosity) fizetjük meg az árát. Például az identitás függvénynek annyi féle definíciója van, ahány típus: $\lambda^{\mathsf{Nat}} x.x : \mathsf{Nat} \Rightarrow \mathsf{Nat}$, $\lambda^{\mathsf{Bool}} x.x : \mathsf{Bool} \Rightarrow \mathsf{Bool}$, $\lambda^{\mathsf{Bool}\Rightarrow\mathsf{Nat}} x.x : (\mathsf{Bool} \Rightarrow \mathsf{Nat}) \Rightarrow (\mathsf{Bool} \Rightarrow \mathsf{Nat})$ stb. Hasonlóan, a függvénykompozíció operátort is külön-külön kell megadnunk minden A_1, A_2, A_3 típushármasra:

$$\lambda^{A_2 \Rightarrow A_3} g.\lambda^{A_1 \Rightarrow A_2} f.\lambda^{A_1} x.g\,(f\,x) : (A_2 \Rightarrow A_3) \Rightarrow (A_1 \Rightarrow A_2) \Rightarrow (A_1 \Rightarrow A_3)$$

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 $\lambda^{\alpha} x.x : \alpha \Rightarrow \alpha$, ahol α-t tetszőleges típussal helyettesíthetjük.

12.1. Szintaxis

$$\begin{aligned} A, A_1, A', ... \in \mathsf{Ty} &::= \alpha \mid A_1 \Rightarrow A_2 \mid \forall \alpha.A \\ t, t_1, t', ... \quad \in \mathsf{Tm} &::= x \mid \lambda^A x.t \mid t\,t' \mid \Lambda\alpha.t \mid t[A] \end{aligned}$$

$\Lambda\alpha.t$ egy polimorf term, mely α típusváltozó bármely konkrét helyettesítésére egységes módon működik. Típusa $\forall\alpha.A$ lesz, ahol A-ban α kötve van ($\forall$ aritása (Tyvar.Ty)Ty, első paraméterében egy Tyvar fajtájú változót köt). Egy ilyen típusú polimorf kifejezésre alkalmazni tudunk egy típust a típusalkalmazás $t[A]$ operátorral, aritása (Tm, Ty)Tm.

12.2. Típusrendszer

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

$$\Delta, \Delta', ... \in \mathsf{TCon} ::= \cdot \mid \Delta, \alpha \tag{12.1}$$

Típusváltozók környezetének kényelmes kezeléséhez kellenek az alábbi definíciók.

$$\begin{aligned} dom(\cdot) &:= \{\} \\ dom(\Delta, \alpha) &:= \{\alpha\} \cup dom(\Delta) \end{aligned} \tag{12.2}$$

$$\frac{}{\cdot\ \mathsf{wf}} \tag{12.3}$$

$$\frac{\Delta\ \mathsf{wf} \qquad \alpha \notin dom(\Delta)}{\Delta, \alpha\ \mathsf{wf}} \tag{12.4}$$

$$\frac{\Delta\,\mathsf{wf} \qquad \alpha \notin dom(\Delta)}{\alpha \in \Delta, \alpha} \tag{12.5}$$

$$\frac{\alpha \in \Delta \qquad \alpha \notin dom(\Delta)}{\alpha \in \Delta, \alpha'} \tag{12.6}$$

A jólformált típus ítélet levezetési szabályai.

$$\frac{\alpha \in \Delta}{\Delta \vdash \alpha} \tag{12.7}$$

$$\frac{\Delta \vdash A_1 \qquad \Delta \vdash A_2}{\Delta \vdash A_1 \Rightarrow A_2} \tag{12.8}$$

$$\frac{\Delta, \alpha \vdash A}{\Delta \vdash \forall \alpha.A} \tag{12.9}$$

Kifejezések típusozási ítélete mostantól hivatkozik egy típuskörnyezetre is, tehát $\Delta\,\Gamma \vdash t : A$ alakú.

$$\frac{(x : A) \in \Gamma}{\Delta\,\Gamma \vdash x : A} \tag{12.10}$$

$$\frac{\Delta\,\Gamma, x : A_1 \vdash t : A_2}{\Delta\,\Gamma \vdash \lambda^{A_1} x.t : A_1 \Rightarrow A_2} \tag{12.11}$$

$$\frac{\Delta\,\Gamma \vdash t : A_1 \Rightarrow A_2 \qquad \Delta\,\Gamma \vdash t : A_1}{\Delta\,\Gamma \vdash t\,t_1 : A_2} \tag{12.12}$$

$$\frac{\Delta, \alpha\,\Gamma \vdash t : A}{\Delta\,\Gamma \vdash \Lambda\alpha.t : \forall \alpha.A} \tag{12.13}$$

$$\frac{\Delta\,\Gamma \vdash t : \forall \alpha.A \qquad \Delta \vdash A_1}{\Delta\,\Gamma \vdash t[A_1] : A[\alpha \mapsto A_1]} \tag{12.14}$$

A polimorf identitás-függvény most már megadható a következőképp:

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

Ezt használhatjuk többféle típusra (ehhez a példához a fenti nyelvet kibővítjük Nat, Bool típusokkal és let-tel):

$$\mathsf{let}\ \Lambda\alpha.\lambda^{\alpha} x.x\ \mathsf{in}\ I.I[\mathsf{Bool}]\,(\mathsf{isZero}\,(I[\mathsf{Nat}]\,(\mathsf{num}\,1)))$$

A polimorf függvénykompozíciót a következőképp adjuk meg.

$$\begin{aligned} &\Lambda\alpha_1.\Lambda\alpha_2.\Lambda\alpha_3.\lambda^{\alpha_2 \Rightarrow \alpha_3} g.\lambda^{\alpha_1 \Rightarrow \alpha_2} f.\lambda^{\alpha_1} x.g\,(f\,x) \\ &\quad : \forall\alpha_1.\forall\alpha_2.\forall\alpha_3.(\alpha_2 \Rightarrow \alpha_3) \Rightarrow (\alpha_1 \Rightarrow \alpha_2) \Rightarrow (\alpha_1 \Rightarrow \alpha_3) \end{aligned}$$

12.3. Operációs szemantika

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

$$\frac{}{\lambda^A x.t\ \mathsf{val}} \tag{12.15}$$

$$\frac{}{\Lambda\alpha.t\ \mathsf{val}} \tag{12.16}$$

$$\frac{[t_1\ \mathsf{val}]}{(\lambda^A\, x.e)\, t_1 \longmapsto t[x \mapsto t_1]} \tag{12.17}$$

$$\frac{t \longmapsto t'}{t\, t_1 \longmapsto t'\, t_1} \tag{12.18}$$

$$\left[\frac{t\ \mathsf{val} \qquad t_1 \longmapsto t_1'}{t\, t_1 \longmapsto t\, t_1'}\right] \tag{12.19}$$

$$\frac{}{(\Lambda\, \alpha.t)[A_1] \longmapsto t[\alpha \mapsto A_1]} \tag{12.20}$$

$$\frac{t \longmapsto t'}{t[A_1] \longmapsto t'[A_1]} \tag{12.21}$$

A haladás és típusmegőrzés tétele bebizonyítható.

12.4. Absztrakció

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

empty	$: A_s$	létrehoz egy üres vermet
push	$: \mathsf{Nat} \Rightarrow A_s \Rightarrow A_s$	verembe rakás
pop	$: A_s \Rightarrow (\mathsf{Opt}\,\mathsf{Nat} \times A_s)$	verem legfelső elemének kiszedése
$\mathsf{isEmpty}$	$: A_s \Rightarrow \mathsf{Bool}$	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 következőképp adható meg.

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

Ezt úgy is szokták hívni, hogy ez egy interface-szel van paraméterezve. Az α jelöli a verem implementációjának a típusát, a kifejezés következő paramétere maga az implementáció, majd következik magának a kifejezésnek a típusa, amely

egyszerűen egy ilyen verem. A következő kifejezés annyit csinál, hogy létrehoz egy üres vermet:

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

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

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

Hány féle érték van az alábbi típusokban?

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

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

$$\cdot\vdash\Lambda\alpha.\lambda^{\alpha}x.x:\forall\alpha.\alpha\Rightarrow\alpha$$

Írhatunk más kifejezést is, például a $\Lambda\alpha.\lambda^{\alpha}x.(\lambda^{\alpha}y.y)\,x$ kifejezés is ugyanilyen típusú lesz, és ez a kifejezés ugyanúgy viselkedik, mint az előző: ugyanarra a bemenetre ugyanazt a kimenetet adja. Ezért azt mondjuk, hogy $\forall\alpha.\alpha\Rightarrow\alpha$-nak

viselkedés szerint csak egyféle eleme van.

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

12.1. Tétel (Parametricitás $t : \forall\alpha.\alpha \Rightarrow \alpha$-ra)**.** *Tegyük fel, hogy van egy R relációnk A_1 és A_2 típusú kifejezések között, mely nem különböztet meg egymásba átírható kifejezéseket, tehát ha $R\,t_1\,t_2$ és $t_1 \longmapsto^* t_1'$ és $t_2 \longmapsto^* t_2'$, akkor $R\,t_1'\,t_2'$. Ekkor, ha $R\,t_1\,t_2$, akkor $R\,(t[A_1]\,t_1)\,(t[A_2]\,t_2)$.*

Ezt a tételt felhasználva beláthatjuk, hogy egy $\forall\alpha.\alpha \Rightarrow \alpha$ típusú t az identitás kell, hogy legyen. Például, ha $A_1 = \mathsf{Nat}$, akkor belátjuk, hogy $t[\mathsf{Nat}]\,(\mathsf{num}\,0) \longmapsto^* * \,\mathsf{num}\,0$. Ehhez úgy adjuk meg az R relációt, hogy $R\,t_1\,t_1 = (t_1 \longmapsto^* \mathsf{num}\,0)$ (az R nem veszi figyelembe a második paraméterét). Ekkor igaz, hogy $R\,(\mathsf{num}\,0)\,t_2$, mert ez azt jelenti, hogy $\mathsf{num}\,0 \longmapsto^* \mathsf{num}\,0$, és emiatt igaz, hogy

$$R\,(t[\mathsf{Nat}]\,(\mathsf{num}\,0))\,(t[A_2]\,t_2),$$

tehát $(t[\mathsf{Nat}]\,(\mathsf{num}\,0)) \longmapsto^* \mathsf{num}\,0$.

Hasonló parametricitás (relációk megtartása) tételek bizonyíthatók az összes többi típusra.

12.4.1. További információ

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

13. Altípus

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

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

$$\frac{\Gamma \vdash t : \mathsf{Nat} \qquad \mathsf{Nat} \leq \mathsf{Int}}{\Gamma \vdash t : \mathsf{Int}}$$

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

További példák, halmazelméleti magyarázattal:

bal $\leq$ jobb	a bal kifejezve a jobbbal
$\mathsf{Nat} \leq \mathsf{Int}$	$\{x \in \mathsf{Int}, x \geq 0\}$
$\mathsf{Int} + \mathsf{String} \leq \mathsf{Int} + \mathsf{Bool} + \mathsf{String}$	$\{x \in \mathsf{Int} + \mathsf{Bool} + \mathsf{String}, x \neq \mathsf{inj}_2(\mathsf{inj}_1\, b)\}$
$\mathsf{Nat} \times \mathsf{String} \leq \mathsf{Int} \times \mathsf{String}$	$\{x \in \mathsf{Int} \times \mathsf{String}, \mathsf{proj}_1\, x \geq 0\}$
$\mathsf{Nat} + \mathsf{String} \leq \mathsf{Int} + \mathsf{String}$	$\{x \in \mathsf{Int} + \mathsf{String}, x = \mathsf{inj}_1\, y$ esetén $y \geq 0\}$
$\mathsf{Int} \times (\mathsf{Bool} \times \mathsf{String}) \leq \mathsf{Int} \times \mathsf{String}$	
$\mathsf{String} \leq \mathsf{String}$	

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

$$\frac{\Gamma \vdash t : A \qquad A \leq A'}{\Gamma \vdash t : A'} \tag{13.1}$$

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.

$$\frac{\{j_1, ..., j_m\} \subset \{i_1, ..., i_n\}}{\Pi(i_1 \hookrightarrow A_{i_1}, ..., i_n \hookrightarrow A_{i_n}) \leq \Pi(j_1 \hookrightarrow A_{j_1}, ..., j_m \hookrightarrow A_{j_m})} \tag{13.2}$$

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

$$\frac{\{i_1, ..., i_n\} \subset \{j_1, ..., j_m\}}{\Sigma(i_1 \hookrightarrow A_{i_1}, ..., i_n \hookrightarrow A_{i_n}) \leq \Sigma(j_1 \hookrightarrow A_{j_1}, ..., j_m \hookrightarrow A_{j_m})} \tag{13.3}$$

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

$$\frac{}{A \leq \mathsf{Unit}} \tag{13.4}$$

Az Empty típus pedig minden más típus altípusa, mert nincs egy konstruktora sem, ezért az összes konstruktorára tudjuk alkalmazni bármely más típus eliminátorát. Az alábbi szabályt 13.1-gyel kombinálva megkapjuk a Empty típus 7.3 eliminációs szabályát.

$$\frac{}{\mathsf{Empty} \leq A} \tag{13.5}$$

Minden függvény, mely egy A_2-beli kifejezést ad vissza, visszaad egy A_2'-beli kifejejezést is, feltéve, hogy $A_2 \leq A_2'$.

$$\frac{A_2 \leq A_2'}{(A_1 \Rightarrow A_2) \leq (A_1 \Rightarrow A_2')} \tag{13.6}$$

Egy $A_1 \Rightarrow A_2$-beli függvény bármely A_1 bemenetre működik, így egy A_1'-beli bemenetre is működik, ha $A_1' \leq A_1$.

$$\frac{A_1' \leq A_1}{(A_1 \Rightarrow A_2) \leq (A_1' \Rightarrow A_2)} \tag{13.7}$$

A kettőt kombinálva a következő szabályt kapjuk:

$$\frac{A_1' \leq A_1 \qquad A_2 \leq A_2'}{(A_1 \Rightarrow A_2) \leq (A_1' \Rightarrow A_2')} \tag{13.8}$$

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).

$$\frac{A_1 \leq A_1' \qquad A_2 \leq A_2'}{A_1 \times A_2 \leq A_1' \times A_2'} \tag{13.9}$$

$$\frac{A_1 \leq A_1' \qquad A_2 \leq A_2'}{A_1 + A_2 \leq A_1' + A_2'} \tag{13.10}$$

Altípusok esetén a típusozás unicitása már nem teljesül, de a haladás és a típusmegőrzés igen. A típuskikövetkeztetést úgy kell megfogalmazni, hogy figyelembe vesszük az altípusokat.

14. Összefoglalás

A különböző típusrendszerekről az alábbiakat láttuk be (zárójelben megadjuk azokat a helyeket, ahol kimondtuk és/vagy bizonyítottuk az adott tulajdonságokat):

- nem triviális (2.8. és 4.1. lemmák),
- unicitás (2.9. és 4.2. lemmák),
- típuskikövetkeztetés (2.10. és 4.3. lemmák),
- környezet permutálhatósága (4.5. lemma),
- gyengítés (4.6. lemma),
- helyettesítési lemma (4.7. lemma),
- dekompozíció (4.8. lemma).

Az operációs szemantikákról az alábbi tulajdonságokat vizsgáltuk:

- nincs olyan t, hogy $t\,\mathsf{val}$ és $t \longmapsto t'$ (2.20 lemma),
- determinisztikus (2.21. lemma)
- nincs végtelen hosszú átírás (2.25. lemma, 9.7. lemma)

A kettő kapcsolatáról a következőket láttuk be:

- típusmegőrzés (2.28. és 4.16. tételek),
- haladás (2.29. és 4.17. tételek).

15. Megoldások

2.1. feladat megoldása.

$$(\mathsf{Tm}, \mathsf{Tm})\mathsf{Tm}$$
$$(\mathsf{Tm})\mathsf{Tm}$$
$$(\mathsf{Tm}, \mathsf{Tm})\mathsf{Tm}$$

2.2. feladat megoldása.

a) igen

b) igen

c) igen

d) nem

e) igen

f) nem

g) igen

2.3. feladat megoldása.

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

2.4. feladat.

2.5. feladat megoldása.

$$\begin{aligned}
&\mathit{eval} \in \mathsf{Tm} \to \mathbb{N}\\
&\mathit{eval}(\mathsf{num}\, n) := n\\
&\mathit{eval}(t + t') := \mathit{eval}(t) + \mathit{eval}(t')\\
&\mathit{eval}(t * t') := \mathit{eval}(t) * \mathit{eval}(t')
\end{aligned}$$

2.6. feladat egy lehetséges megoldása.

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

2.7. feladat részleges megoldása.

$$
\begin{aligned}
&igen \\
&igen \\
&igen \\
&nem \\
&nem \\
&nem
\end{aligned}
$$

2.11. feladat.
2.13. feladat.
2.14. feladat.
2.15. feladat megoldása.

$$
\begin{aligned}
&\mathsf{num}\, 0 + \mathsf{false} \\
&\mathsf{isZero}\, \mathsf{false} \\
&\mathsf{if}\, \mathsf{num}\, 1\, \mathsf{then}\, \mathsf{true}\, \mathsf{else}\, \mathsf{false}
\end{aligned}
$$

2.16. feladat megoldása.
Nem akad el.
2.17. feladat.
2.18. feladat.

$$\mathsf{if}\, t\, \mathsf{then}\, \mathsf{false}\, \mathsf{else}\, \mathsf{true}$$

2.19. feladat.

2.22. feladat részleges megoldása.

$$\frac{t_2 \longmapsto t_2'}{t_1 + t_2 \longmapsto t_1 + t_2'} \tag{15.1}$$

$$\frac{t_2\,\mathsf{val} \qquad t_1 \longmapsto t_1'}{t_1 + t_2 \longmapsto t_1' + t_2} \tag{15.2}$$

2.23. feladat.

Nem, mert a $(\mathsf{num}\,1 + \mathsf{num}\,2) + (\mathsf{num}\,1 + \mathsf{num}\,2)$-re két különböző egy lépéses átírást tudunk alkalmazni, melyeknek más az eredménye: az egyiknek $\mathsf{num}\,3 + {} + (\mathsf{num}\,1 + \mathsf{num}\,2)$, a másiknak $(\mathsf{num}\,1 + \mathsf{num}\,2) + \mathsf{num}\,3$.

2.24. feladat részleges megoldása.

Nem.

2.26. feladat.

2.31. feladat.

Ilyen a 2.16. feladatban látott $\mathsf{if\,true\,then\,num}\,1\,\mathsf{else}\,(\mathsf{num}\,1 + \mathsf{false})$ kifejezés.

2.32. feladat.

2.33. feladat.

3.1. feladat.

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

$$\overline{x}\,\overline{y}$$

$$\overline{x}\,(\lambda \underset{1}{x}.\overline{y}\,\underset{1}{x})$$

$$\overline{x}\,(\lambda \underset{1}{x}.\overline{x'}\,(\underset{1}{x}\,(\lambda \underset{2}{x'}.\underset{2}{x'}\,\underset{1}{x})))$$

$$\overline{x}\,(\lambda \underset{1}{x}.\underset{1}{x}\,(\underset{1}{x}\,(\lambda \underset{2}{x}.\underset{2}{x}\,\underset{2}{x})))$$

$$\overline{x}\,(\lambda \underset{1}{x}.(\underset{1}{x}\,(\lambda \underset{2}{x'}.\overline{x''}\,\underset{2}{x'})))\,(\overline{x'}\,(\lambda \underset{3}{x'}.\underset{3}{x'}\,\overline{x''}))$$

3.2. feladat megoldása.

$$
\begin{aligned}
x\,y &\neq x\,z \\
x\,y\,z &\neq x\,(y\,z) \\
x\,(y\,z) &\neq (x\,y)\,z \\
x\,(\lambda x.y\,x) &= x\,(\lambda x'.y\,x') \\
x\,(\lambda x.y\,x) &\neq x\,(\lambda x.y\,x') \\
x\,(\lambda x.x'\,(x\,(\lambda x'.x'\,x))) &\neq x\,(\lambda x'.x\,(x'\,(\lambda x.x\,x'))) \\
x\,(\lambda x.x'\,(x\,(\lambda x'.x'\,x))) &= x\,(\lambda x''.x'\,(x''\,(\lambda x.x\,x''))) \\
x\,(\lambda x.x\,(x\,(\lambda x.x\,x))) &= x\,(\lambda x'.x'\,(x'\,(\lambda x'.x'\,x')))
\end{aligned}
$$

3.3. feladat megoldása.

$$
\begin{aligned}
(x\,(\lambda x.x\,x))[x \mapsto z] &= z\,(\lambda x.x\,x) \\
(x'\,(\lambda x.x'\,x))[x' \mapsto z] &= (z\,(\lambda x.z\,x)) \\
(x'\,(\lambda x.x'\,x'))[x' \mapsto x] &= (x\,(\lambda z.x\,x)) \\
(x'\,(\lambda x.x'\,x))[x' \mapsto x] &= (x\,(\lambda z.x\,z))
\end{aligned}
$$

3.4. feladat.
3.6. feladat.
3.7. feladat.
3.8. feladat.
3.9. feladat.
3.10. feladat.
3.11. feladat.
4.4. feladat.
4.9. feladat részleges megoldása.

– igen

– igen

– nem

– nem

4.10. feladat.
4.11. feladat.
4.12. feladat megoldása.

$$\mathsf{let}\ \mathsf{num}\ 1 + \mathsf{num}\ 2\ \mathsf{in}\ x.x + x$$

4.13. feladat megoldása.

$$\mathsf{let}\,\mathsf{num}\,1 + \mathsf{num}\,2\,\mathsf{in}\,x.\mathsf{num}\,1$$

4.14. feladat megoldása.

$$\mathsf{let}\,\mathsf{num}\,1 + \mathsf{num}\,2\,\mathsf{in}\,x.x$$

4.15. feladat.

5.1. feladat megoldása.

$$\mathsf{Nat}$$
$$\mathsf{Bool}$$
$$\mathsf{Nat} \Rightarrow \mathsf{Nat}$$
$$\mathsf{Nat} \Rightarrow \mathsf{Bool}$$
$$\mathsf{Bool} \Rightarrow \mathsf{Nat}$$
$$\mathsf{Bool} \Rightarrow \mathsf{Bool}$$
$$(\mathsf{Nat} \Rightarrow \mathsf{Nat}) \Rightarrow \mathsf{Nat}$$
$$\mathsf{Nat} \Rightarrow (\mathsf{Nat} \Rightarrow \mathsf{Nat})$$
$$(\mathsf{Nat} \Rightarrow \mathsf{Nat}) \Rightarrow (\mathsf{Nat} \Rightarrow \mathsf{Nat})$$
$$(\mathsf{Nat} \Rightarrow (\mathsf{Nat} \Rightarrow \mathsf{Nat})) \Rightarrow \mathsf{Nat}$$

5.2. feladat megoldása.

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{\cdot\,\mathsf{wf}}\ 4.4 \qquad x \notin \{\}}{\cdot, x : \mathsf{Nat}\,\mathsf{wf}}\ 4.5}{\cdot, x : \mathsf{Nat} \vdash \mathsf{num}\,2 : \mathsf{Nat}}\ 4.8 \qquad \dfrac{\dfrac{\overline{\cdot\,\mathsf{wf}}\ 4.4 \qquad x \notin \{\}}{(x : \mathsf{Nat}) \in \cdot, x : \mathsf{Nat}}\ 4.6}{\cdot, x : \mathsf{Nat} \vdash x : \mathsf{Nat}}\ 4.14}{\cdot, x : \mathsf{Nat} \vdash \mathsf{num}\,2 + x : \mathsf{Nat}}\ 4.9}{\cdot \vdash \lambda^{\mathsf{Nat}} x.\mathsf{num}\,2 + x : \mathsf{Nat} \Rightarrow \mathsf{Nat}}\ 5.3$$

5.3. feladat.

5.4. feladat megoldása.

$$\lambda^{\mathsf{Nat}} x.\lambda^{\mathsf{Bool}} y.\mathsf{if}\ y\ \mathsf{then}\ x\ \mathsf{else}\ \mathsf{num}\,0$$

5.5. feladat megoldása.

$$\lambda^{\mathsf{Bool}\Rightarrow\mathsf{Bool}} x.\lambda^{\mathsf{Bool}} y.x\,(x\,(x\,y))$$

5.6. feladat.

5.7. feladat.

5.8. feladat.

6.1. feladat.

Mindkettő esetben egy ilyen van, $\langle\langle \mathsf{tt}, \mathsf{tt}\rangle, \mathsf{tt}\rangle$, illetve $\langle \mathsf{tt}, \langle \mathsf{tt}, \mathsf{tt}\rangle\rangle$.

6.2. feladat.

– érték szerint, mohón

$$\begin{aligned}
&(\lambda^{\mathsf{Nat}} x.\langle x + \mathsf{num}\,3, x + \mathsf{num}\,4\rangle)\,(\mathsf{num}\,1 + \mathsf{num}\,2)\\
\overset{5.7}{\longmapsto}&(\lambda^{\mathsf{Nat}} x.\langle x + \mathsf{num}\,3, x + \mathsf{num}\,4\rangle)\,(\mathsf{num}\,3)\\
\overset{5.8}{\longmapsto}&\langle \mathsf{num}\,3 + \mathsf{num}\,3, \mathsf{num}\,3 + \mathsf{num}\,4\rangle\\
\overset{6.9}{\longmapsto}&\langle \mathsf{num}\,6, \mathsf{num}\,3 + \mathsf{num}\,4\rangle\\
\overset{6.10}{\longmapsto}&\langle \mathsf{num}\,6, \mathsf{num}\,7\rangle
\end{aligned}$$

– név szerint, mohón

$$\begin{aligned}
&(\lambda^{\mathsf{Nat}} x.\langle x + \mathsf{num}\,3, x + \mathsf{num}\,4\rangle)\,(\mathsf{num}\,1 + \mathsf{num}\,2)\\
\overset{5.8}{\longmapsto}&\langle (\mathsf{num}\,1 + \mathsf{num}\,2) + \mathsf{num}\,3, (\mathsf{num}\,1 + \mathsf{num}\,2) + \mathsf{num}\,4\rangle\\
\overset{6.9}{\longmapsto}&\langle \mathsf{num}\,3 + \mathsf{num}\,3, (\mathsf{num}\,1 + \mathsf{num}\,2) + \mathsf{num}\,4\rangle\\
\overset{6.9}{\longmapsto}&\langle \mathsf{num}\,6, (\mathsf{num}\,1 + \mathsf{num}\,2) + \mathsf{num}\,4\rangle\\
\overset{6.10}{\longmapsto}&\langle \mathsf{num}\,6, \mathsf{num}\,3 + \mathsf{num}\,4\rangle\\
\overset{6.10}{\longmapsto}&\langle \mathsf{num}\,6, \mathsf{num}\,7\rangle
\end{aligned}$$

– érték szerint lustán

$$\begin{aligned}
&(\lambda^{\mathsf{Nat}} x.\langle x + \mathsf{num}\,3, x + \mathsf{num}\,4\rangle)\,(\mathsf{num}\,1 + \mathsf{num}\,2)\\
\overset{5.7}{\longmapsto}&(\lambda^{\mathsf{Nat}} x.\langle x + \mathsf{num}\,3, x + \mathsf{num}\,4\rangle)\,(\mathsf{num}\,3)\\
\overset{5.8}{\longmapsto}&\langle \mathsf{num}\,3 + \mathsf{num}\,3, \mathsf{num}\,3 + \mathsf{num}\,4\rangle
\end{aligned}$$

– név szerint lustán

$$\begin{aligned}
&(\lambda^{\mathsf{Nat}} x.\langle x + \mathsf{num}\,3, x + \mathsf{num}\,4\rangle)\,(\mathsf{num}\,1 + \mathsf{num}\,2)\\
\overset{5.8}{\longmapsto}&\langle (\mathsf{num}\,1 + \mathsf{num}\,2) + \mathsf{num}\,3, (\mathsf{num}\,1 + \mathsf{num}\,2) + \mathsf{num}\,4\rangle
\end{aligned}$$

6.3. feladat.

6.4. feladat megoldása.

$$\lambda^{\mathsf{Nat}\times(\mathsf{Nat}\times\mathsf{Nat})}x.\mathsf{proj}_1\,x + (\mathsf{proj}_1\,(\mathsf{proj}_2\,x) + \mathsf{proj}_2\,(\mathsf{proj}_2\,x))$$

6.5. feladat megoldása.

$$\lambda^{\mathsf{Nat}}x_1.\lambda^{\mathsf{Nat}}x_{12}.\lambda^{\mathsf{Nat}}x_{22}.x_1 + (x_{12} + x_{22})$$

6.6. feladat.

$$\lambda^{\mathsf{Nat}}x.\langle x + \mathsf{num}\,1, x + \mathsf{num}\,2\rangle$$

6.7. feladat.
6.8. feladat.
6.9. feladat.
7.1. feladat.
$\mathsf{Bool} := \mathsf{Unit} + \mathsf{Unit}$
$\mathsf{true} := \mathsf{inj}_1\,\mathsf{tt}$
$\mathsf{false} := \mathsf{inj}_2\,\mathsf{tt}$
$\mathsf{if}\,t\,\mathsf{then}\,t_1\,\mathsf{else}\,t_2 := \mathsf{case}\,t\,x_1.t_1\,x_2.t_2$
7.2. feladat.
7.3. feladat.
7.4. feladat.
7.5. feladat.
7.6. feladat.
7.7. feladat.
7.8. feladat.
7.9. feladat részleges megoldása.

$\mathsf{Bool}\times\mathsf{Unit}$	és Bool	izomorfak
$(\mathsf{Unit}+\mathsf{Unit})+\mathsf{Unit}$	és $\mathsf{Unit}+(\mathsf{Unit}+\mathsf{Unit})$	izomorfak
$(A+A')\times A''$	és $(A\times A'')+(A'\times A'')$	izomorfak
$A+\mathsf{Empty}$	és A	izomorfak
$A\times A'$	és $A'\times A$	izomorfak
$A\times\mathsf{Empty}$	és Empty	izomorfak

7.10. feladat.
7.11. feladat.
8.1. feladat.

1. A következő bizonyítást elnevezzük p-nek.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot, (X \supset (Y \supset Z)) \ni (X \supset (Y \supset Z))}\,8.3}{\cdot, (X \supset (Y \supset Z)), Y \ni (X \supset (Y \supset Z))}\,8.4}{\cdot, (X \supset (Y \supset Z)), Y, X \ni (X \supset (Y \supset Z))}\,8.4}{\cdot, (X \supset (Y \supset Z)), Y, X \vdash (X \supset (Y \supset Z))}\,8.5$$

A kért bizonyítás a következő.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot, (X \supset (Y \supset Z)), Y, X \ni X}\,8.3}{\cdot, (X \supset (Y \supset Z)), Y, X \vdash X}\,8.5 \quad p}{\cdot, (X \supset (Y \supset Z)), Y, X \vdash Y \supset Z}\,8.15 \quad \dfrac{\dfrac{\dfrac{}{\cdot, (X \supset (Y \supset Z)), Y \ni Y}\,8.3}{\cdot, (X \supset (Y \supset Z)), Y, X \ni Y}\,8.4}{\cdot, (X \supset (Y \supset Z)), Y, X \vdash Y}\,8.5}{\cdot, (X \supset (Y \supset Z)), Y, X \vdash Z}\,8.15$$
$$\dfrac{\dfrac{\dfrac{\cdot, (X \supset (Y \supset Z)), Y \vdash (X \supset Z)}{\cdot, (X \supset (Y \supset Z)) \vdash (Y \supset X \supset Z)}\,8.14}{\cdot \vdash (X \supset (Y \supset Z)) \supset (Y \supset X \supset Z)}\,8.14}{}\,8.14$$

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 $(X \supset (X \wedge \top)) \wedge ((X \wedge \wedge \top) \supset X)$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot, X \ni X}\,8.3}{\cdot, X \vdash X}\,8.5 \quad \dfrac{}{\cdot, X \vdash \top}\,8.6}{\cdot, X \vdash (X \wedge \top)}\,8.8}{\cdot \vdash X \supset (X \wedge \top)}\,8.14 \quad \dfrac{\dfrac{\dfrac{}{\cdot, (X \wedge \top) \ni X \wedge \top}\,8.3}{\cdot, (X \wedge \top) \vdash X \wedge \top}\,8.5}{\cdot, (X \wedge \top) \vdash X}\,8.9}{\cdot \vdash ((X \wedge \top) \supset X)}\,8.14$$
$$\cdot \vdash (X \supset (X \wedge \top)) \wedge ((X \wedge \top) \supset X)\,8.8$$

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.

9.4. feladat.

9.5. feladat.

9.6. feladat.

10.1. feladat.

$$\dfrac{\dfrac{\dfrac{}{\alpha.\mathsf{Unit}\ \mathsf{poly}}\ 10.2 \quad \dfrac{}{\alpha.\alpha\ \mathsf{poly}}\ 10.1}{\alpha.\mathsf{Unit}\times\alpha\ \mathsf{poly}}\ 10.3 \quad \dfrac{\dfrac{}{\alpha.\alpha\ \mathsf{poly}}\ 10.1 \quad \dfrac{}{\alpha.\alpha\ \mathsf{poly}}\ 10.1}{\alpha.\alpha\times\alpha\ \mathsf{poly}}\ 10.3}{\alpha.(\mathsf{Unit}\times\alpha)+\alpha\times\alpha\ \mathsf{poly}}\ 10.5 \quad \dfrac{}{\alpha.\mathsf{Empty}\ \mathsf{poly}}\ 10.4}{\alpha.(\mathsf{Empty}+((\mathsf{Unit}\times\alpha)+\alpha\times\alpha)\ \mathsf{poly}}\ 10.5$$

10.2. feladat.

$\mathsf{Empty}+((\mathsf{Unit}\times(\mathsf{Nat}+\mathsf{Bool}))+(\mathsf{Nat}+\mathsf{Bool})\times(\mathsf{Nat}+\mathsf{Bool}))$

10.3. feladat.

Ehhez a feladathoz a következő szabályok szükségesek:

$$\dfrac{}{\alpha.\mathsf{Bool}\ \mathsf{poly}}$$

illetve

$$\dfrac{}{\mathsf{map}^{\alpha.\mathsf{Bool}}\,(x'.t'')\,t\longmapsto t} \qquad (*)$$

megoldás:

$$\begin{aligned}
&\mathsf{map}^{\alpha.\mathsf{Unit}+(\mathsf{Bool}\times\alpha)}\,(x'.\mathsf{suc}\,x')\,(\mathsf{inj}_2\langle\mathsf{true},t\rangle)\\
\overset{10.12}{\longmapsto}\quad&\mathsf{case}\,(\mathsf{inj}_2\langle\mathsf{true},t\rangle)\\
&\quad(x_1.\mathsf{inj}_1\,(\mathsf{map}^{\alpha.\mathsf{Unit}}\,(x'.\mathsf{suc}\,x')\,x_1))\\
&\quad(x_2.\mathsf{inj}_2\,(\mathsf{map}^{\alpha.\mathsf{Bool}\times\alpha}\,(x'.\mathsf{suc}\,x')\,x_2))\\
\overset{7.14}{\longmapsto}\quad&\mathsf{inj}_2\,(\mathsf{map}^{\alpha.\mathsf{Bool}\times\alpha}\,(x'.\mathsf{suc}\,x')\,\langle\mathsf{true},t\rangle)\\
\overset{7.11,10.10}{\longmapsto}\quad&\mathsf{inj}_2\,\langle\mathsf{map}^{\alpha.\mathsf{Bool}}\,(x'.\mathsf{suc}\,x')\ \mathsf{true},\mathsf{map}^{\alpha.\alpha}\,(x'.\mathsf{suc}\,x')\,t\rangle\\
\overset{7.11,6.9,(*)}{\longmapsto}\quad&\mathsf{inj}_2\,\langle\mathsf{true},\mathsf{map}^{\alpha.\alpha}\,(x'.\mathsf{suc}\ x')\,t\rangle\\
\overset{7.11,6.10,10.8}{\longmapsto}\quad&\mathsf{inj}_2\,\langle\mathsf{true},\mathsf{suc}\,t\rangle
\end{aligned}$$

10.4. feladat.

10.5. feladat.

10.6. feladat.

10.7. feladat.

11.1. feladat.

11.3. feladat.

11.4. feladat.

11.5. feladat.

11.6. feladat.

11.7. feladat.

11.8. feladat.

11.9. feladat.

11.10. feladat.

11.11. feladat.

11.12. feladat.

11.13. feladat.

11.14. feladat.

Tárgymutató

Hivatkozások

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[2] Zoltán Csörnyei. *Lambda-kalkulus. A funkcionális programozás alapjai*. Typotex, 2007.

[3] Zoltán Csörnyei. *Fordítóprogramok*. Az informatika alkalmazásai. Typotex, 2009.

[4] Zoltán Csörnyei. *Bevezetés a típusrendszerek elméletébe*. ELTE Eötvös Kiadó, 2012.

[5] Robert Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, New York, NY, USA, 2nd edition, 2016.

[6] Donald E. Knuth. Backus Normal Form vs. Backus Naur Form. *Commun. ACM*, 7(12):735–736, December 1964.

[7] Benjamin C. Pierce. *Basic Category Theory for Computer Scientists*. The MIT Press, 1991.

[8] Benjamin C. Pierce. *Types and Programming Languages*. The MIT Press, 1st edition, 2002.

[9] Andrew M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge University Press, New York, NY, USA, 2013.

[10] The Univalent Foundations Program. Homotopy type theory: Univalent foundations of mathematics. Technical report, Institute for Advanced Study, 2013. Available online: `http://homotopytypetheory.org/book`.

[11] Peter Selinger. Lecture notes on the lambda calculus. *CoRR*, abs/0804.3434, 2008.

[12] The Agda development team. Agda wiki, 2017.