EÖTVÖS LORÁND TUDOMÁNYEGYETEM
INFORMATIKAI KAR

CSÖRNYEI ZOLTÁN

FUNKCIONÁLIS PROGRAMNYELVEK IMPLEMENTÁCIÓJA

III. RÉSZ

TÍPUSELMÉLET

BUDAPEST, 2003

Tartalomjegyzék

III. RÉSZ

A TÍPUSOS λ-KALKULUS

Előszó

Church 1932-ben kidolgozott egy olyan, a függvények és a logika elméletével foglalkozó rendszert, amit a matematika alapjának szánt. Kleene és Rosser azonban 1936-ban megmutatta, hogy ez a rendszer inkonzisztens, így Churchnek az a terve, hogy rendszerével a matematika minden területe leírható legyen, nem valósulhatott meg. De a függvényekkel foglalkozó rész, a λ-kalkulus alkalmasnak bizonyult a kiszámítható függvények modellezésére. Kleene és Rosser 1936-ban bebizonyította, hogy minden rekurzív függvény leírható λ-kalkulussal, sőt mi több, Turing 1937-ben megmutatta, hogy a Turing-kiszámítható függvények osztálya azonos a λ-kalkulussal leírható függvények osztályával. A kiszámítható függvények λ-kalkulusbeli leírásából, a λ-kifejezésekből fejlődött ki az 1950-es években a funkcionális programozás, az első funkcionális programnyelv.

A λ-kalkulus, pontosabban a λ-kalkulus egy variánsának, a kombinátor logikának típusokkal való kiegészítése és a típus fogalmának bevezetése Currytól származik, 1934-ből. A típusos λ-kalkulus pontos kidolgozása az 1960-as, 1970-es években történt, amire természetesen már hatással volt a funkcionális programozás típusfogalma is. A Curry által meghatározott *Curry-típusos λ-kalkulus* elve az, hogy minden λ-kifejezéshez hozzá lehet rendelni a lehetséges típusainak a halmazát. Ez a halmaz lehet üres, tartalmazhat egy, vagy akár végtelen sok lehetséges típust. Egy funkcionális programot típus nélkül is meg lehet írni, és majd a compiler meghatározza azt, hogy vajon milyen típust lehet a λ-kifejezéshez hozzárendelni. Ha a program „helyes”, akkor a típus meghatározható. A típus meghatározását *típuslevezetésnek* nevezzük. Ez az „*implicit típusosság*”, és ezt először az ML-nyelvcsaládban alkalmazták.

1940-ben Church is foglalkozott a típus bevezetésével, és rendszerében minden λ-kifejezéshez egyértelműen rendelt hozzá egy típust. Ez azt jelenti, hogy a felhasználónak, azaz a programozónak minden λ-kifejezéshez egyértelműen meg kell adnia a λ-kifejezés típusát. Ekkor a compilerben a típusvizsgálat legtöbbször csak egy egyszerű típusellenőrzés, és a compilernek nem kell típuslevezetést végeznie. Ez a λ-kalkulus a *Church-típusos λ-kalkulus*, lényege az „*explicit típusosság*”. Ez a típusosság elsősorban az imperatív programnyelvekre jellemző.

1. FEJEZET

Bevezetés

1.1. A típus kialakulása

(A típus csak egy illúzió...)

Kezdetben minden egyforma, egységes volt, típus nélküli volt a világ. Ez máris egy érdekes ellentmondás, mert mit jelent az, hogy nincs semminek sem típusa? Azt, hogy mindennek azonos a típusa, azaz, hogy van egy típus, de csak egy típus van, és mindennek ez az egy típus a típusa.

Ilyen homogén rendszer például a számítógép memóriája, ahol csak bitek vannak, minden bit felveheti a 0 vagy az 1 értéket, és a bitekre csak a logikai alapműveleteket értelmezzük. Az ember egyes bitkombinációkhoz már ismert fogalmakat rendel, bizonyos bitekre azt mondja, hogy ez a nulla, másokra azt, hogy ez az 1, 2, vagy a 49 természetes szám. Egészen más bitcsoportra meg azt, hogy ez az összeadás művelete, egy másik meg a szorzásé. Ugyanakkor a 65 szám bitkombinációja lesz az „A" betű, a 66 a „B", két ilyen bitkombináció egymás mellé írva pedig egy karakterstringet jelent. Ezután ha újabb, hosszabb, nulla bitekből álló bitsorozatot jelölünk ki, akkor ez lesz a nulla lebegőpontos szám, mások meg az 1, 2 lebegőpontos számok.

Vagy például ilyen a λ-kalkulus, ahol mindent egy-egy λ-kifejezéssel írunk le, és csak a kifejezések átalakítására adunk meg szabályokat. A λ-kalkulusban, a bitkombinációkhoz hasonlóan, egy kifejezéshez hozzárendelhetjük a nulla számot, egy másikhoz az 1-et, a harmadikhoz a 2-t, egy E kifejezésre pedig azt mondhatjuk, hogy ez az összeadás művelete, mert ha n és m két természetes számnak megfeleltetett λ-kifejezés, az $E\ n\ m$ applikáció éppen az $n + m$-nek megfeleltetett kifejezést adja eredményül. Találunk egészen más kifejezéseket is, ezekre is megtehetjük ezeket a hozzárendeléseket, és ekkor egy másik „számrendszert" kapunk. Vagy kiválaszthatunk három λ-kifejezést úgy, hogy az elsőre a másodikat alkalmazva a harmadikat, az elsőre a harmadikat applikálva a másodikat kapjuk eredményül, és ekkor azt mondjuk, hogy az első a *not*, függvény, a második a *true*, a harmadik a *false* érték. Ez utóbbi kettőhöz rendelt neveket akár fel is cserélhetjük.

A homogén rendszerből kiválasztott elemek közül bizonyosak tehát a kijelölt műveletek szempontjából hasonlóan viselkednek. Az azonos viselkedésű elemeket egy halmazba fogjuk össze, és ez a halmaz alkotja a típust. *A típus tehát értékek halmaza.* Ezért azt mondhatjuk, hogy „egy érték egy típus egy eleme", „egy érték egy típushoz tartozik". A típus értékhalmazának egy megkülönböztető nevet adhatunk, és azt is mondhatjuk, hogy a halmaz elemei ilyen típusúak.

1.1.1. Példa. *(Típusok)*

A true és false értékek alkotják a *Bool* típust.

A $0, 1, 2\ldots$ számok a *Nat* nevű típus elemei. □

Látszik tehát, hogy a típus csak illúzió, a típusnélküli világot csak mi képzeljük típusosnak. Ez a típusos világ azonban megvédi az alatta levő nem típusos világot a hibás, vagy a nem jó használattól. A típusrendszer feltörése, például egy *Integer* típusú szám *Pointer*-ként való használata, gyakran nagyon kellemetlen következményekkel jár.

1.2. A típus szerepe

A programnyelvekben típust alapvetően két célból használunk,

- bizonyos programtulajdonságok leírásának eszközeként,
- és a programok biztonságos működésének szavatolására.

Típusok használatával nem csak egyes programelemek struktúráját tudjuk leírni, például azt, hogy egy változó karaktersorozatokból álló tömb, hanem egyes programelemek működését is, például megadhatjuk azt, hogy egy függvény inputja egy egész szám, eredménye pedig logikai értékek listája. Az ilyen tulajdonságok leírhatósága a program jobb struktúráltságán kívül a jobb megértést és a könnyebb programírást is lehetővé teszi, és ezáltal jobb programok írását eredményezi.

A típusok használata a fordítás hatékonyságát is növeli, bár a típusok fordításban való felhasználásának a következő hatását is figyelembe kell venni. A polimorfikus típusok lehetővé teszik azt, hogy ugyanazt a programkódot különböző típusú adatokra használjuk, az absztrakt adattípus használata pedig a kód struktúrálását segíti. De mivel a típussal kapcsolatos vizsgálatokat fordítási időben kell végezni, a fordítóprogram csak véges időben eldönthető problémák vizsgálatával foglalkozhat, azaz egy típusrendszernek olyannak kell lennie, hogy benne a nem-terminálás ne fordulhasson elő.

A Curry-Howard izomorfizmus lehetővé teszi azt, hogy a matematikai logika módszereit és eredményeit a típuselméletben is felhasználhassuk. Ezzel a témakörrel egy későbbi fejezetben foglalkozunk.

Azokat a nyelveket, amelyekben a változóknak típusuk van, *típusos nyelveknek* nevezzük. Egy nyelv lehet típusos úgy, hogy a típus a nyelv szintaxisának része, ezeket a nyelveket *explicit típusos nyelveknek* nevezzük, de egy lehet típusos úgy is, hogy a típusfogalom a szintaxisban egyáltalán nem fordul elő. Ezek az *implicit típusos nyelvek*.

Explicit típusos nyelv például a Haskell, Clean, implicit típusos nyelvre példa az ML. Implicit típusos nyelvek esetén a típus bevezetése és a típushozzárendelés a fordítóprogram feladata.

Azok a programnyelvek, amelyekben a változók értékkészletének nem adunk meg halmazokat, azaz a változókhoz nem rendelünk típust, a *nem típusos nyelvek* közé tartoznak. Ezekben a nyelvekben egy műveletet tetszőleges argumentumra elő lehet írni, és az eredmény is tetszőleges „típusú" értéket felvehet, az eredmény lehet például egy eltérülés, vagy más run-time hiba is. A típusnélküli λ-kalkulus a nem típusos nyelvek egy olyan tipikus példája, ahol a műveleteknek ez a hibára utaló eredménye nem fordul elő, itt csak egy „művelet" van, az applikáció, és ha az applikáció első tagja nem függvény, akkor azt mondjuk, hogy a kifejezés nem változik, azaz a kifejezés normál formában van.

2. FEJEZET

Formális típusrendszerek

2.1. Formális matematikai rendszer

A *formális matematikai rendszer* szimbólumok és a rájuk alkalmazható szabályok rendszere.

2.1.1. Definíció. Formális matematikai rendszer:

A Φ formális matematikai rendszer egy $(\Sigma, \mathcal{F}, \mathcal{A}, \mathcal{R})$ négyes, ahol

- *Σ egy ábécé, azaz az alapszimbólumok halmaza. Az alapszimbólumokból álló véges sorozatokat formuláknak nevezzük.*
- *$\mathcal{F}$ a nyelvtani szabályok halmaza. Azokat a formulákat, amelyek megfelelnek a nyelvtani szabályoknak, jól formált formuláknak nevezzük.*
- *$\mathcal{A}$ az axiómák halmaza. A jól formált formulák egy speciális halmazáról feltesszük, hogy jelentéssel bírnak, ezek az axiómák.*
- *$\mathcal{R}$ a levezetési szabályok halmaza. A levezetési szabályok felhasználásával az axiómáktól különböző formulákról bizonyíthatjuk be, hogy van jelentésük.*

2.1.2. Példa. *(Egy G grammatika, mint formális matematikai rendszer)*
Egy $G = (T, N, S, \mathcal{P})$ grammatika által meghatározott mondatformák leírhatók egy formális matematikai rendszerrel. A formális matematikai rendszer Σ ábécéje legyen a $T \cup N$ halmaz, az $\mathcal{F}$ nyelvtani szabályok halmaza legyen üres, a grammatika S szimbóluma legyen az $\mathcal{A}$ egyetlen axiómája, és a $\mathcal{P}$ halmaz elemei, azaz a grammatika levezetési szabályai legyenek a formális matematikai rendszer $\mathcal{R}$-beli levezetési szabályai. Tehát $\Phi = (T \cup N, \emptyset, S, \mathcal{P})$.

Ebben a rendszerben az ábécé elemeiből képzett minden szimbólumsorozat jól formált formula, és az S-ből a $\mathcal{P}$-beli szabályok alkalmazásával levezethető szimbólumsorozatoknak, a mondatformáknak van jelentésük. □

A következő pontban látni fogjuk, hogy a formális típusrendszer is egy speciális formális matematikai rendszer.

2.2. Formális típusrendszerek

Egy formális típusrendszer megadásához először meg kell adni a kifejezések *szintaktikáját*. A formális típusrendszer kifejezései két komponensből állnak, λ-kifejezésből és a λ-kifejezés típusát leíró típuskifejezésből. Ennek megfelelően a szintaktika is két részből áll, az egyik rész a típusok kifejezéseinek, a másik rész a λ-kifejezéseknek a szintaktikáját írja le. Mindkét szintaktika környezetfüggetlen grammatikával írható le.

A továbbiakban a *típuskifejezéseket*, vagy röviden a *típusokat* az $A, B, \ldots$ betűkkel, a λ-kifejezéseket továbbra is az $E, F, \ldots$ betűkkel jelöljük. A *típusváltozók* jelölésére az $\alpha, \beta, \ldots$ betűket, a λ-kifejezések változóinak jelölésére most is az $x, y, \ldots$ betűket használjuk.

A formális típusrendszer kifejezéseit $E\!:\!A$ alakban írjuk le, ahol tehát E egy λ-kifejezés, és A az E típusát leíró típuskifejezés.

A szintaktikusan helyes kifejezéseknek és a szintaktikusan helyes típusoknak külön nevet is adhatunk. A kifejezések nevét $\mathsf{sans\ serif}$ betűtípussal írjuk, a típusok neve nagybetűvel kezdődik, és leírásukra az *Italic* betűtípust használjuk. A kifejezések azonosságát és a típusazonosságot az $\equiv$ jel jelöli.

2.2.1. Definíció. Jól formált λ-kifejezés:

Egy λ-kifejezés jól formált, ha szintaktikusan helyes, azaz megfelel a λ-kifejezések szintaktikájának leírásában megadott szintaktikai szabályoknak.

2.2.2. Definíció. Jól formált típus:

Egy típus jól formált, ha szintaktikusan helyes, azaz megfelel a típus szintaktiktikájának leírásában megadott szintaktikai szabályoknak.

Ha egy jól formált kifejezésben csak jól formált típusok szerepelnek, ez még nem garantálja azt, hogy a kifejezés a típus szempontjából hibátlan. Egy jól formált kifejezésre a típusok jól formáltságának megkövetelése nem elég, ennek az az oka, hogy a helyes típus leírására nem elegendőek a környezetfüggetlen szabályok. Az, hogy egy kifejezés a típus szempontjából mikor helyes, nem írható le környezetfüggetlen gramatikákkal. A típus helyességét csak további vizsgálatokkal tudjuk eldönteni, erre szolgál a típuskörnyezet és a következtetési szabályok halmaza.

Egy formális típusrendszerben *következtetéseket* adhatunk meg. Egy következtetés alakja

$\Gamma \vdash \mathcal{I}$,

ahol Γ a *statikus típuskörnyezet*, és $\mathcal{I}$ a Γ-ból adódó *állítás*.

A statikus típuskörnyezet elnevezésből a „statikus” jelzőt el is hagyhatjuk.

2.2.3. Definíció. A típuskörnyezet szintaktikája:

$$\begin{array}{lcl} \langle \textit{típuskörnyezet} \rangle & ::= & \emptyset \\ & | & \langle \textit{típuskörnyezet} \rangle \, , \langle \textit{változó} \rangle : \langle \textit{típus} \rangle \end{array}$$

A típuskörnyezetekre még a következő tulajdonságokat is megköveteljük:

- *Ha x_i egy A_i típusú változó ($1 \leq i \leq n$), akkor egy típuskörnyezet az $x_i\!:\!A_i$ párokból felépített*

  $x_1\!:\!A_1, x_2\!:\!A_2, \ldots, x_n\!:\!A_n$

  lista.
- *Ha $i \neq j$, akkor $x_i \neq x_j$, azaz a típuskörnyezetben egy változó csak egyszer szerepelhet, ami azt jelenti, hogy egy változónak csak egy típusa lehet.*

A Γ típuskörnyezetet arra fogjuk használni, hogy λ-kifejezések szabad változóinak típusait meghatározzuk, a szabad változók típusait majd innen fogjuk „kiolvasni”.

Ha a Γ típuskörnyezet egy párt sem tartalmaz, akkor a típuskörnyezetre az $\emptyset$ jelet használjuk. A Γ-ban szereplő változók halmazát $dom(\Gamma)$-val jelöljük. A Γ-t egy olyan függvénynek is tekinthetjük, amelyre ha $x_i : A_i \in \Gamma$, akkor $\Gamma(x_i) = A_i$.

2.2.4. Definíció. Jól formált típuskörnyezet:

Egy típuskörnyezet jól formált, ha szintaktikusan helyes, azaz megfelel a 4.1.2. definícióban megadott szabályoknak.

Ha a Γ típuskörnyezet jól formált, akkor ezt a

$\Gamma \;\vdash \mathit{wf}$

következtetéssel jelöljük. A $\emptyset$ üres típuskörnyezet nyilvánvalóan mindig jól formált, azaz $\emptyset \;\vdash \mathit{wf}$. Ha a Γ típuskörnyezetben az A típus jól formált, akkor ezt a tulajdonságot a

$\Gamma \;\vdash A$

jelöli. Számunkra most a

$\Gamma \;\vdash E\!:\!A$

alakú *következtetések* a fontosak. A $\Gamma \vdash E : A$ következtetés azt mondja ki, hogy figyelembe véve az E szabad változóinak a Γ típuskörnyezetben levő típusait, az E kifejezés típusa A.

Megjegyezzük, hogy egy jól formált Γ típuskörnyezet statikus, azaz ha Γ-ban egy x változóra $x:A$, akkor ez a későbbiekben nem változik meg. Még akkor sem, ha ez a típus beépül egy λ-kifejezésbe, és a típus kikerül a Γ-ból. Nem lehet a későbbiekben az $x:B$ típust megadni, ha $A \neq B$. Például a $\lambda x.\lambda x.y$ kifejezésben nem lehet a külső x típusa A, a belső x típusa pedig B, azaz ha y típusa C, akkor nem lehet a kifejezés típusa $A \to (B \to C)$.

A formális típusrendszerben egy következtetés lehet *érvényes*, vagy *érvénytelen*. Egy következtetés érvényességének bizonyítására a típusrendszer *szabályai* szolgálnak. Egy szabály

$$\frac{\Gamma_1 \vdash \mathcal{I}_1 \quad \ldots \Gamma_n \vdash \mathcal{I}_n}{\Gamma \vdash \mathcal{I}} \quad [\text{SZABÁLYNÉV}]$$

alakú, ahol a vízszintes vonal fölött a *feltétel* következtetései, a vonal alatt a feltételekből származtatott *következmény* következtetése van. Egy szabály a feltételben szereplő következtetések érvényességet nem vizsgálja, a szabály azt mondja ki, hogy ha a feltétel mindegyik következtetése érvényes, akkor a következmény következtetése is érvényes.

A szabályoknak nevet is adhatunk, és ha szükséges, a feltételek mellett jobboldalon még a szabály alkalmazásának feltételeit is megadhatjuk.

Ha egy szabályban a feltételek halmaza üres, azaz a vonal felett nincs egy állítás sem, akkor a következtetés mindig érvényes. Az ilyen következtetéseket *axiómának* nevezzük.

Mint majd a 3.1. pontban látni fogjuk, a szabályokat a jól formált típuskörnyezet leírására is használhatjuk.

2.2.5. Példa. *(Az üres típuskörnyezet mindig jól formált)*

$$\frac{}{\emptyset \vdash \mathit{wf}} \quad [\text{ENV } \emptyset]$$

A szabály azt írja le, hogy az üres típuskörnyezet minden esetben jól formált. Ez a szabály a majd definiálandó következtetési rendszerekben axiómaként fog szerepelni. □

A szabályokat *típuslevezetések* készítésére használjuk, úgy, hogy egy *következtetési fát* vagy más néven *levezetési fát* építünk fel. A szabályokat egymáshoz kapcsoljuk, az SI szabály következményének következtetése az SJ szabály feltételének egy következtetéséhez kapcsolható, ha a két következtetés megegyezik.

2.2.6. Példa. *(Levezetési fák)*

$$\dfrac{\dfrac{\dfrac{K_{11}}{K_{12}}\,[\mathrm{S1}]}{K_{22}}\,[\mathrm{S2}]}{K_{32}}\,[\mathrm{S3}] \qquad \dfrac{\dfrac{\dfrac{K_{41}}{K_{42}}\,[\mathrm{S4}]}{K_{52}}\,[\mathrm{S5}] \quad \dfrac{K_{61}}{K_{62}}\,[\mathrm{S6}]}{K_{72}}\,[\mathrm{S7}] \qquad \square$$

2.2.7. Definíció. Érvényes következtetés:

Azt mondjuk, hogy a $\Gamma \vdash E : A$ *következtetés érvényes, ha létezik olyan típuslevezetés, ahol a következtetés a típuslevezetéshez tartozó következtetési fa gyökérpontja.*

2.2.8. Definíció. Jól típusozott kifejezés:

Ha egy típusrendszerben a $\Gamma \vdash E : A$ *következtetés érvényes, akkor azt mondjuk, hogy az* E *kifejezés jól típusozott.*

A $\Gamma \vdash E : A$ következtetés érvényessége tehát azt jelenti, hogy az E kifejezés a típus szempontjából is helyes, és a következtetés azt is megadja, hogy az E kifejezés típusa A. Könnyen belátható, hogy ha egy kifejezés jól típusozott, akkor a kifejezés minden részkifejezése is jól típusozott.

A jól típusozott kifejezést *típusozható* kifejezésnek is nevezhetjük.

Ezek után most már megadhatjuk a *formális típusrendszer* definícióját.

2.2.9. Definíció. Formális típusrendszer:

Egy $\mathcal{T}$ *formális típusrendszer egy* $(\mathcal{S}, \mathcal{J}, \mathcal{R})$ *hármas, ahol*

- $\mathcal{S}$ *a típusrendszer szintaxisa, a típusok és a kifejezések szintaktikáját adja meg,*
- $\mathcal{J}$ *a következtetések formáit írja le,*
- $\mathcal{R}$ *a következtetések érvényességének bizonyítására szolgáló szabályok halmaza.*

2.2.10. Példa. *(Egy formális típusrendszer és a formális matematikai rendszer)*

A $(\mathcal{S}, \mathcal{I}, \mathcal{R})$ formális típusrendszer egy olyan formális matematikai rendszer, amelyben a típusokat és kifejezéseket alkotó, azaz az $\mathcal{S}$ szintaktika leírásában szereplő szimbólumok és jelek halmaza a formális rendszer ábécéje, és a típusrendszer $\mathcal{S}$ szintaxisa a formális rendszer nyelvtani szabályainak halmaza. Az $\mathcal{R}$ szabályhalmaz axiómái a formális rendszer axiómái, és az $\mathcal{R}$ nem-axióma szabályai lesznek a formális rendszer szabályai.

Tehát a típusrendszer jól formált kifejezései a formális matematikai rendszer jól formált formulái, és a jól típusozott kifejezések a formális matematikai rendszer jelentéssel bíró formulái. $\square$

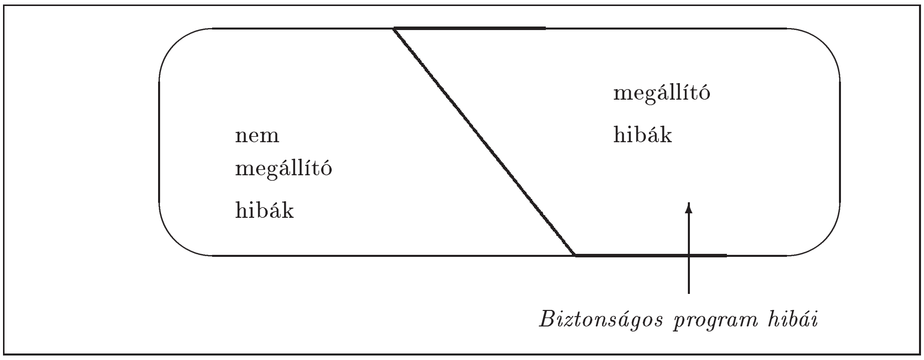
2.3. A típus és a biztonság

A típusok alkalmazásának alapvető célja az, hogy a program futásakor keletkező végrehajtási hibákat megelőzzük. Egy végrehajtási hiba jelentkezése esetén

- vagy azonnal abbamarad a program végrehajtása, mert egy *kivétel* lép fel (ezeket a hibákat hívjuk *megállító hibáknak*),
- vagy a program fut tovább, és egyáltalán nem, vagy esetleg később jelentkezik valamilyen hiba (ezeket a végrehajtási hibákat hívjuk *nem megállító hibáknak*).

Megállító hiba például a nullával való osztás, vagy egy negatív számból történő gyökvonás. Ezek a hibák úgy jellemezhetők, hogy egy műveletet nem a megfelelő argumentumra hajtunk végre. Ide tartoznak a hardver korlátok túllépéséből származó hibák is, például a nem létező utasításkód végrehajtása, vagy nem létező memória címzése.

Nem megállító hiba egy tömbelem olvasása esetén az indexhatár túllépése, azaz a tömb előtti vagy utáni memóriaterületről történő olvasás, vagy rossz címre, például egy adatmezőbe történő vezérlésátadás. A nem megállító hibák típusvizsgálattal kiszűrhetők, mivel ezeket a hibákat a típusokra vonatkozó szabályok megsértése okozza.



2.1. ábra. Biztonságos program

Azt mondjuk, hogy egy program *biztonságos*, ha nem okoz nem megállító hibát, azaz a program minden hiba esetén megáll. Azokat a programnyelveket, amelyekben minden futtatható program biztonságos, *biztonságos nyelveknek* nevezzük.

A típusos nyelvek egy része úgy biztosítja a biztonságot, hogy már a fordítóprogram fordítási időben visszautasítja a nem biztonságos programokat. A nem típusos nyelveknél a biztonság elérése csak a futási időben végzett ellenőrzéssel valósítható meg.

Természetesen nem minden nyelv biztonságos, és a biztonság független attól, hogy a nyelv típusos-e. A 2.2. ábrán néhány programnyelvet osztályozunk ezen szempontok szerint.

	típusos nyelv	*nem típusos nyelv*
biztonságos	ML, Haskell, Clean	LISP
nem biztonságos	C	assembly

2.2. ábra. Programnyelvek osztályozása

Egy nyelv biztonsága a programok fejlesztési és futtatási idejének arányát határozza meg. A biztonság eléréséhez szükséges vizsgáltok elhagyását legtöbbször azzal magyarázzák, hogy ezzel a hatékonyságot növelik, mivel biztonságos típusos nyelveknél a programfejlesztési idő nőhet meg, hiszen a fordítóprogram csak a biztonságos programot fogadja el, a biztonságos nem típusos nyelvek esetén pedig a run-time időben elvégzendő ellenőrzések a program futási idejét növelhetik meg jelentősen.

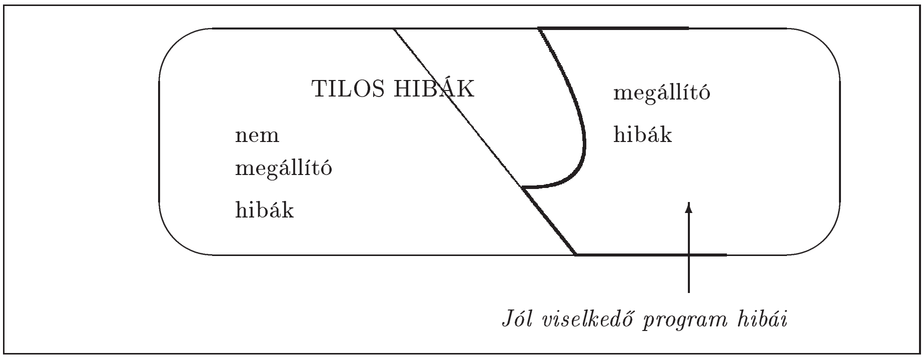
A biztonság ugyanakkor sok szempontból a költségek megtakarítását is jelenti, például egy biztonságos programban a hibakeresésre fordított idő minimális, vagy például lehetővé válik a hulladékgyűjtés, ami a program méretét is csökkentheti.

A programnyelvekben a nem megállító hibákat és a megállító hibák egy részét *tilos hibáknak* nevezzük. Tilos hibák azok a megállító hibák, ahol a művelet argumentuma nem megfelelő, mint például a `not 3` kifejezésben.

Egy programot *jól viselkedő programnak* nevezünk, ha nem okoz tilos hibát.

Nyilvánvaló, hogy minden jól viselkedő program biztonságos, de ez fordítva nem áll fenn. A típusrendszerek jól viselkedő programok előállítását tűzik ki célul, ez a biztonság vizsgálatát is magába foglalja.

Azokat a nyelveket, amelyekben minden futtatható program jól viselkedő, *erősen ellenőrzött nyelveknek* nevezzük. Így egy erősen ellenőrzött nyelvű programban nem lehetnek nem megállító hibák, és nem lehetnek olyan megállító



2.3. ábra. Jól viselkedő program

hibák, amelyek tilosak.

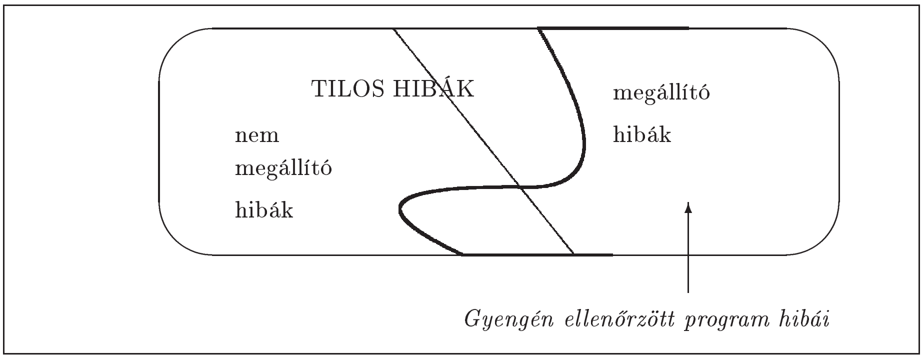
Ha egy nyelv típusos, akkor a jól viselkedés vizsgálata statikusan, azaz fordítási időben végezhető. Ezt az ellenőrzést *típusellenőrzésnek*, a típusellenőrzéssel ellenőrzött programot *jól típusozott programnak* nevezzük. Ha a típusellenőrzés hibát jelez, akkor a program *rosszul típusozott program.* A rosszul típusozott program nem garantálja a jól viselkedést, futásakor tilos hibák előfordulhatnak.

A nem típusos nyelvek is lehetnek jól viselkedőek, még akkor is, ha egyáltalán nem tartozik hozzájuk típusellenőrzés. A szükséges vizsgálatok ekkor csak futási időben végezhetők el, ezért ezek az ellenőrzések egy tilos hiba esetén, még nem megállító hiba esetén is, megállítják a program futását.

Erősen ellenőrzött, típusos nyelv az ML, és erősen ellenőrzött, de nem típusos nyelv a LISP.

Az lenne az optimális, ha a jól viselkedő programok minden megállító hibát kiszűrnének, azaz minden megállító hiba a tilos hibák közé tartozna. Sajnos ez nincs így, sőt a gyakorlatban az ellenőrzés még a nem megállító hibák egy részére sem terjed ki. Az ilyen nyelveket *gyengén ellenőrzött nyelveknek* nevezzük.

Gyengén ellenőrzött nyelv például a Pascal és a C, nem vizsgált leállító hiba a C-ben például a pointer-aritmetika hibás használata miatt jelentkező pointerhiba.



2.4. ábra. Gyengén ellenőrzött nyelv programja

2.4. Típushelyesség

Láttuk, hogy típusos rendszerekben a jól viselkedést a típusellenőrzés vizsgálta, és ha az ellenőrzés nem talált hibát, akkor a programot jól típusozott programnak neveztük. A jól típusozott program tehát nem okoz tilos hibát.

A formális típusrendszerekben is definiáltunk egy jól típusozott fogalmat, azt mondtuk, hogy egy E kifejezés jól típusozott egy Γ típuskörnyezetben, ha létezik olyan A típus, melyre $\Gamma \vdash E{:}A$.

Mivel a funkcionális programok a λ-kalkulus kifejezéseire fordíthatók le, a két jól típusozott fogalom kapcsolatát bizonyítani kell, azaz meg kell mutatni, hogy ha egy E kifejezés jól típusozott, akkor a hozzátartozó program nem okoz tilos hibát.

Vizsgáljuk R. Milnernek[?] azt a közismert állítását, hogy

- *egy jól típusozott program nem fut rosszul.* (1)

Ez az állítás, pontosabban ennek az alábbiakban finomított változata a típushelyesség alapja. A típushelyesség ugyanis a következőket mondja ki. A típusrendszer akkor helyes, ha minden olyan program, amelyre a típusrendszerrel megállapítható a program jól típusozottsága, nem működik hibásan, azaz nem fut rosszul.

A jól típusozottság fogalma egyértelmű, most a „nem fut rosszul” kifejezést elemezzük.

A típusrendszer a programot fordítási időben, azaz statikusan vizsgálja. Ha egy program inputja egy természetes szám, outputja is egy természetes szám, akkor a típusrendszer megállapítja, hogy a program egy

$Num \rightarrow Num$

leképezést végez, de nem foglalkozik azzal, hogy a program például négyzetre emelést vagy éppen négyzetgyökvonást hajt végre. A típusrendszer tehát a programokat csak típusuk szerint különbözteti meg.

A típusrendszer nem a program szemantikáját vizsgálja, ezért meg kell különböztetnünk a program *run-time szemantikáját* és a program *típus szemantikáját*, és látjuk, hogy a típus szemantika nem a run-time szemantika másolata. Mivel a típustrendszer egyáltalán nem foglalkozik a run-time szemantikával, a fenti idézetet így kell módosítanunk:

- *egy jól típusozott program futása közben nem jelez run-time hibát.* (2)

Run-time hibák a végrehajtó rendszer által detektált hibák, amelyek például a nem megfelelő függvény-argumentumok, vagy egy aritmetikai művelet nem megfelelő típusú operandusai esetén jelentkeznek. Nyilvánvalóan nem nevezhetjük run-time hibának azt, amikor a program run-time szemantikája nem egyezik meg a programíró szándékával.

A fenti (2) állítás csak egy irányban igaz, azaz ha egy program futása közben nem jelentkezik run-time hiba, ez még nem jelenti azt, hogy a program jól típusozott. A *Megállás Problémája* tétel szerint van olyan program, amire ez utóbbi állítás teljesül, van olyan run-time hiba jelzése nélkül futó program, aminek a jól típusozottsága nem bizonyítható.

Ezek után próbáljuk megfogalmazni a típushelyességet:

- $\forall P,$ *ha* $\Gamma \vdash P{:}A,$ *akkor* $P \Rightarrow E$ *esetén* $\Gamma \vdash E{:}A,$ (3)

ahol a $P \Rightarrow E$ azt jelenti, hogy a P program futásának az eredménye E.

A (2) állítás egy kicsit mást mond, mint ez a megfogalmazás, ez a leírás azt állítja, hogy a jól típusozott programnak mindig van egy E eredménye, és az eredmény típusa azonos a program típusával. Ez a megfogalmazás erősebb, mint a (2) állítás, de célszerű a típushelyességnek ezt választani, mert ezt könnyebb bizonyítani.

A típushelyesség tehát két dologtól függ: a típusokat meghatározó típusrendszertől, és a $\Rightarrow$ levezetést meghatározó redukciós rendszertől. Tehát azt mondhatjuk, hogy egy redukciós rendszert figyelembe véve a típusrendszer helyes, ha egy program típusa megegyezik a program redukciójával kapott eredmény típusával.

A típushelyesség fenti megfogalmazása azonban még nem tökéletes. Például nyilvánvalóan

$(x{:}Num/y{:}Num){:}Num,$

azaz a program jól típusozott, de mi történik, ha $y = 0$? Ez nyilván megsérti a típushelyesség állítását, hiszen a futási időben a végrehajtó rendszer run-time hibát fog jelezni.

Mi legyen a hibajelzés?

- Lehetne például egy számérték, mondjuk -1. Ez azonban nem lehet jó megoldás, hiszen ekkor a $2 + (1/0)$ kifejezésnek lenne értéke, pontosan 1.
- Lehetne egy végtelen ciklusba való belépés. De ekkor hogyan különböztessük meg a run-time hibákat egymástól, és ezeket a run-time hibákat a tényleges végtelen ciklustól?
- Egy kivétel generálása tűnik a jó megoldásnak, ez egyértelműen jelezheti a hibát, és lehetőséget ad a programozónak arra, hogy a kivételhez megadja a saját kivételkezelő programját.

Ha a program redukálásának valamelyik lépésében kivételt generálhatunk, akkor ez azt jelenti, hogy a program, mint függvény, egy parciális függvény, és a „nincs értelmezve" fogalomnak itt a kivétel generálása felel meg. Például az x/y függvény egy parciális függvény, mert nincs értelmezve az $y = 0$ pontban.

Elemezve a programok végrehajtását, előre megadhatjuk a kivételeknek egy $\mathcal{X}$ halmazát, ebbe azok a kivételek tartoznak, amelyek a programfüggvények parciális tulajdonságából adódnak.

Ezzel az $\mathcal{X}$ halmazzal megadhatjuk a típushelyesség újabb megfogalmazását:

- $\forall P$, *ha* $\Gamma \vdash P{:}A$, *akkor*
  - $P \Rightarrow E$ esetén $\Gamma \vdash E{:}A$, (4)
  - *vagy a P redukálása folyamán egy $\mathcal{X}$-beli kivétel lép fel.*

Nem foglalkoztunk még azokkal a problémákkal, hogy a P program típusának meghatározásakor végtelen ciklus léphet fel, és hogy nem feltétlenül véges a P redukálásainak száma sem.

Az első probléma megoldása egyszerű, ha a P típusa nem határozható meg véges lépésben, akkor a P nem jól típusozható. A végtelen redukciós sorozat azonban természetesnek mondható, a rekurzív függvények definiálásával, a rekurzió alkalmazásával könnyen kaphatunk olyan programokat, amelyek nem redukálhatók véges lépésben.

Ezért a típushelyesség fogalmát tovább kell finomítanunk, és ezzel megkapjuk a végső megfogalmazást.

2.4.1. Definíció. Típushelyesség

A típusrendszert típusosan helyesnek nevezzük, ha

- $\forall P$ *programra* $\Gamma \vdash P : A$*, és*
- *a* P *redukálásának terminálása esetén*
  - *ha* $P \Rightarrow E$*, akkor* $\Gamma \vdash E : A$*,*
  - *vagy a* P *redukálása folyamán egy* $\mathcal{X}$*-beli kivétel lép fel.*

Egy típusrendszer helyességét bizonyítani kell. A típusrendszer helyességének állítását a *típushelyesség tételének* nevezzük, és ha az állítás igaz, akkor azt is mondhatjuk, hogy a típusrendszerben a típushelyesség teljesül.

Mint majd látni fogjuk, a továbbiakban vizsgálandó formális típusrendszerek mindegyike helyes típusrendszer lesz.

3. FEJEZET

Az elsőrendű típusos λ-kalkulus

Ebben a fejezetben a legegyszerűbb típusos λ-kalkulussal foglalkozunk, az *elsőrendű típusos λ-kalkulussal.* Ennek a λ-kalkulusnak szemléletükben különböző két fő tárgyalási módszere van, a módszerek elnevezése az első publikálójukra utal.

- A *Curry-típusos λ-kalkulus* 1934-ből származik [?]. Curry 1934-ben a típusos kombinátor logika elméletét dolgozta ki, és erre alapozva készítették el később a Curry-típusos λ-kalkulust. A kalkulust *implicit* típusos rendszernek nevezzük, mivel nem kell a kifejezések megadásakor a típusokat megadni. Későbbi felhasználási területe a *típuslevezetés* lett, ez a programnyelvek implementációiban lehetővé teszi azt, hogy a forrásnyelvű programban nem kell típusokat megadni, a programnyelv fordítóprogramja határozza meg a kifejezések típusát. Az implicit típusos programnyelvre a klasszikus példa az ML.
- Church típusos λ-kalkulusról szóló cikke [?] 1940-ben jelent meg. A *Church-típusos λ-kalkulus* jellemzője az, hogy minden típusnélküli kifejezésnek egyértelmű, a kifejezés definiálásakor megadott vagy levezethető típusa van. Ezt a rendszert *explicit* típusos λ-kalkulusnak nevezzük. Az implementációban ennek a λ-kalkulusnak az alkalmazása a *típusellenőrzést* jelenti, ilyen tipikus programnyelv a Pascal vagy az Algol.

A gyakorlatban a Curry-típusos λ-kalkulust „típuslevezetéses λ-kalkulusnak” nevezik, és Church rendszere kapta „a típusos λ-kalkulus” elnevezést.

A továbbiakban mi is elsősorban a Church-típusos rendszerrel foglalkozunk, de majd tanulmányozzuk a típuslevezetés módszereit is.

3.1. Az elsőrendű Church-típusos $\boldsymbol{\lambda}$-kalkulus

Az elsőrendű Church-típusos λ-kalkulust F_1 *típusrendszernek*, röviden F_1 vagy F *rendszernek* nevezzük. Ebben a rendszerben a szembetűnő változás a típusnélküli λ-kalkulushoz viszonyítva az, hogy a λ-absztrakciókban a változók típusát is jelöljük:

$\lambda x{:}A\,.\,E,$

azaz a λ-kifejezésekben a kötött változó típust kap, az x változó típusa ebben a λ-kifejezésben A. A programnyelvekben gyakran az E típusát is meg kell adni, az F_1 rendszerben erre nem lesz szükség.

Egy függvény típusát $A \to B$-vel jelöljük, ahol A és B típusok, $\to$ a *típuskonstruktor*. Az A a függvény argumentumának, B a függvény értékének a típusa. Az $A \to B$ típust *függvénytípusnak* is nevezhetjük. Ahhoz, hogy ilyen típusokat fel tudjunk építeni, szükségünk van kezdeti típusokra, azaz az *alaptípusok* halmazára. Egy ilyen alaptípushalmaz lehet például a $\{Bool,\ Nat\}$. Az alaptípusok jó megválasztása azért fontos, mert egy adott alaptípushalmazra felépített F_1 rendszerben a típusokban csak az alaptípushalmaz elemei szerepelhetnek, az alaptípusok halmazának bővítése egy másik F_1 rendszert eredményez.

3.2. Az $\boldsymbol{F_1}$ rendszer szintaxisa

Az 1.1.1. definícióban látható, hogy a típusnélküli λ-kalkulus kifejezéseit egy egyszerű, környezetfüggetlen szintaxissal le tudtuk írni. Most megadjuk az F_1 rendszer szintaxisát.

3.2.1. Definíció. Az $\boldsymbol{F_1}$ rendszer szintaxisa:

$$\begin{array}{lcl} <\textit{típus}> & ::= & <\textit{alaptípus}> \\ & | & (<\textit{típus}> \to <\textit{típus}>) \\ <\lambda\textit{-kifejezés}> & ::= & <\textit{változó}> \\ & | & (\lambda\ <\textit{változó}> :<\textit{típus}>\ .\ <\lambda\textit{-kifejezés}>) \\ & | & (<\lambda\textit{-kifejezés}>\ <\lambda\textit{-kifejezés}>) \end{array}$$

A legkülső zárójelpárt elhagyhatjuk.

Régebbi publikációkban az $A \to B$ függvénytípusra az $\mathsf{F}AB$, (AB), A^B, B^A vagy a (BA) jelölést is használták.

A fenti definícióból látható, hogy a típusnélküli λ-kalkulus leírása a típusokra vonatkozó szintaktikus szabályokkal bővült. Az F_1 rendszer szintaxisa röviden a

$\mathbb{T} = \mathbb{U} \mid \mathbb{T} \to \mathbb{T}$,

$\Lambda_{\mathbb{T}} = V \mid \lambda x{:}\mathbb{T}\,.\,\Lambda_{\mathbb{T}} \mid (\Lambda_{\mathbb{T}}\Lambda_{\mathbb{T}})$

alakban is írható, ahol $\mathbb{T}$ a típusokat, $\mathbb{U}$ az alaptípusokat, V a változókat, $\Lambda_{\mathbb{T}}$ a λ-kifejezéseket jelöli.

Megjegyezzük, hogy az E egy λ-kifejezést jelöl, és $E{:}A$ egy típust. Az $E{:}A$ azt jelenti, hogy az E kifejezés típusa A, azaz az E az A típus egy eleme. Ha ezt a tartalmazást akarjuk hangsúlyozni, akkor az $E \in A$ jelölést is alkalmazhatjuk.

Az $E{:}A$ típusban E-nek, azaz a λ-kifejezésnek mindig nagyobb a *precedenciája*, mint az A-nak, ezt figyelembe véve $(EF){:}A \equiv EF{:}A$, és $(\lambda x{:}A\,.\,E){:}B \equiv \lambda x{:}A\,.\,E{:}B$.

A λ-absztrakció természetesen most is *jobbasszociatív*, az applikáció pedig *balasszociatív.*

A függvénytípus *jobbasszociatív*, ennek előnyét a következő példában mutatjuk meg.

3.2.2. Példa. *(A függvénytípus jobbasszociatív)*
A Church-számokon értelmezett $add(x,y)$ összeadás függvény a $(Nat \times Nat) \to Nat$ leképezést valósítja meg. A currying módszert alkalmazva láttuk (**??** példa), hogy

$add(x,y) = add_x(y) = (f(x))(y)$.

Legyen az add_x függvény típusa ρ, ekkor az f függvény típusa $Nat \to \rho$. Az add_x függvény típusa nyilvánvalóan $Nat \to Nat$. Így az $add(x,y)$ típusa

$Nat \to (Nat \to Nat)$,

ami a jobbasszociatívitás miatt

$Nat \to Nat \to Nat$

alakban is írható. □

A típusokban és a λ-kifejezésekben a legkülső zárójelpárt és az asszociatívitások miatti redundáns zárójelpárokat elhagyhatjuk.

A 3.2.1 definíció a szintaktikus szabályokat írja le, de nem minden szintaktikusan helyes, azaz jól formált λ-kifejezés „jó" egy jól formált típus szempontjából.

3.2.3. Példa. *(A típus szempontjából hibás λ-kifejezés)*
Ha A egy alaptípus, tehát nyilvánvalóan jól formált, akkor a $\lambda x{:}A\,.\,xy$ kifejezés szintaktikusan helyes. A típus szempontjából azonban biztosan hibás, mivel az xy törzsből az következik, hogy az x-nek $A \to B$ típusúnak kell lennie,

valamilyen B típusra. Az x típusa tehát függvénytípus, azaz nem lehet alaptípus. □

A fenti példából látszik, hogy az elsőrendű típusos λ-kalkulusban a típusosan helyes kifejezések megadásához nem elegendő a szintaxis megadása.

3.3. Az F_1 rendszer következtetései és szabályai

Az F_1 rendszer szintaktikus szabályait már megadtuk, most adjuk meg a típusrendszer további leírását.

Az F_1 rendszerben három fajta következtetés van, az első és a második egy típuskörnyezet és egy típus jól formáltságát mondja ki, a harmadik egy jól formált kifejezés típusát adja meg.

3.3.1. Definíció. Az F_1 rendszer következtetései:

$\Gamma \vdash \mathit{wf}$	Γ jól formált környezet
$\Gamma \vdash A$	Γ-ban az A jól formált típus
$\Gamma \vdash E{:}A$	Γ-ban az E kifejezés típusa A

Az F_1 rendszerben a következtetések érvényességének bizonyítására a következő szabályokat kell használni.

3.3.2. Definíció. Az F_1 rendszer szabályai:

A környezetre vonatkozó szabályok:

$$\frac{}{\emptyset \vdash \mathit{wf}} \quad [\textsc{Env}\ \emptyset]$$

$$\frac{\Gamma \vdash A \quad x \notin \mathit{dom}(\Gamma)}{\Gamma, x{:}A \vdash \mathit{wf}} \quad [\textsc{Env}\ x]$$

A típusra vonatkozó szabályok:

$$\frac{\Gamma \vdash \mathit{wf} \quad A \in \mathit{alaptípus}}{\Gamma \vdash A} \quad [\textsc{Type Const}]$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \to B} \quad [\textsc{Type Arrow}]$$

Kifejezés típusára vonatkozó szabályok:

$$\frac{\Gamma', x{:}A, \Gamma'' \vdash \mathit{wf}}{\Gamma', x{:}A, \Gamma'' \vdash x{:}A} \quad [\textsc{Val}\ x]$$

$$\frac{\Gamma, x{:}A \vdash E{:}B}{\Gamma \vdash \lambda x{:}A\,.\,E{:}A \to B} \quad [\textsc{Val Fun}]$$

$$\frac{\Gamma \vdash E{:}A \to B \quad \Gamma \vdash F{:}A}{\Gamma \vdash EF{:}B} \quad [\textsc{Val Appl}]$$

Az F_1 rendszerben csak egy axióma van, az Env $\emptyset$.

A 3.3.2. definícióból az is látható, hogy a típusra vonatkozó szabályok (Type Const, Type Arrow) csak a szintaktikai szabályokat ismétlik meg, így, ha egy kifejezés A típusa jól formált, azaz szintaktikusan helyes, akkor a $\Gamma \vdash A$ következtetés mindig érvényes. Az ilyen következtetés érvényességének vizsgálatára tehát az F_1 rendszerben még nincs szükség. Az ilyen következtetések érvényességét majd csak a magasabb rendű típusrendszerekben kell vizsgálni. Így tehát az F_1 rendszerben az alapvető feladat a kifejezések jól típusozottságának a vizsgálata.

A Val x szabály feltételei a jól formáltságon kívül azt jelentik, hogy $\Gamma(x) = A$, ezért a szabály egy gyengébb alakja a következő:

$$\frac{\Gamma(x) = A}{\Gamma \vdash x{:}A}$$

Ebből a szabályból az is látható, hogy egy x szabad változó csak akkor lesz jól típusozott, ha típusa a Γ típuskörnyezetben benne van, azaz az x változó A típusára $x : A \in \Gamma$.

A Val Fun szabály feltételében levő következtetés típuskörnyezetében $\Gamma, x :$

A szerepel, ami azt jelenti, hogy az x nincs benne a Γ-ban, vagyis $x \notin dom(\Gamma)$. Látható, hogy a feltétel típuskörnyezetéből az $x\!:\!A$ beépül a λ-absztrakcióba.

A VAL FUN szabályt, mivel egy λ-kifejezésbe egy típust épít be, „típusbevezetésnek”, a VAL APPL szabályt pedig, mivel függvénytípusból egy egyszerű típust ad eredményül, „típuseliminációnak” is nevezzük.

3.3.3. Példa. *(Az* id_Nat $\equiv \lambda x\!:\!Nat\,.\,x$ *kifejezés típusának meghatározása)*
Az alaptípusok halmaza legyen $K = \{Bool, Nat\}$. Ekkor

$$\dfrac{\dfrac{\dfrac{\dfrac{\emptyset \vdash wf \quad Nat \in K}{\emptyset \vdash Nat} \qquad x \notin dom(\emptyset)}{x\!:\!Nat \vdash wf}}{x\!:\!Nat \vdash x\!:\!Nat}}{\emptyset \vdash (\lambda x\!:\!Nat\,.\,x)\!:\!Nat \to Nat}$$

□

3.3.4. Példa. *(A $\lambda x\!:\!Bool \to Nat\,.\,\lambda y\!:\!Bool\,.\,xy$ kifejezés típusának meghatározása)*
Az alaptípusok halmaza legyen $K = \{Bool, Nat\}$. Ekkor

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\emptyset \vdash wf} \quad Bool \in K}{\emptyset \vdash Bool} \qquad \dfrac{\dfrac{}{\emptyset \vdash wf} \quad Nat \in K}{\emptyset \vdash Nat}}{\emptyset \vdash Bool \to Nat} \qquad x \notin dom(\emptyset)}{x\!:\!Bool \to Nat \vdash wf}$$

Hasonlóan, az y-ra:

$$\dfrac{\dfrac{x\!:\!Bool \to Nat \vdash wf \qquad Bool \in K}{x\!:\!Bool \to Nat \vdash Bool} \qquad y \notin dom(x\!:\!Bool \to Nat)}{x\!:\!Bool \to Nat,\ y\!:\!Bool \vdash wf}$$

Ebből az x típusa:

$$\dfrac{x\!:\!Bool \to Nat,\ y\!:\!Bool \vdash wf}{x\!:\!Bool \to Nat,\ y\!:\!Bool \vdash x\!:\!Bool \to Nat}$$

Hasonlóan, az y típusa:

$$\dfrac{x\!:\!Bool \to Nat,\ y\!:\!Bool \vdash wf}{x\!:\!Bool \to Nat,\ y\!:\!Bool \vdash y\!:\!Bool}$$

Ezekből pedig xy-ra:

$$\frac{\begin{array}{l} x\!:\!Bool \to Nat,\ y\!:\!Bool\ \vdash\ x\!:\!Bool \to Nat \\ x\!:\!Bool \to Nat,\ y\!:\!Bool\ \vdash\ y\!:\!Bool \end{array}}{x\!:\!Bool \to Nat,\ y\!:\!Bool\ \vdash xy\!:\!Nat}$$

Most határozzuk meg az absztrakciók típusát:

$$\cfrac{\cfrac{x\!:\!Bool \to Nat,\ y\!:\!Bool\ \vdash xy\!:\!Nat}{x\!:\!Bool \to Nat\ \vdash (\lambda y\!:\!Bool\,.\,xy)\!:\!Bool \to Nat}}{\emptyset\ \vdash (\lambda x\!:\!Bool \to Nat\,.\,\lambda y\!:\!Bool\,.\,xy)\!:\!(Bool \to Nat) \to Bool \to Nat}$$

Mivel a

$$\emptyset\ \vdash (\lambda x\!:\!Bool \to Nat\,.\,\lambda y\!:\!Bool\,.\,xy)\!:\!(Bool \to Nat) \to Bool \to Nat$$

következtetés a levezetési fa gyökérpontja és a következtetés érvényes, a $\lambda x : Bool \to Nat\,.\,\lambda y\!:\!Bool\,.\,xy$ kifejezés jól típusozott, és a kifejezés típusa $(Bool \to Nat) \to Bool \to Nat$. □

3.3.5. Példa. *(A $\lambda x\!:\!Bool \to Nat\,.\,xy$ kifejezés típusának meghatározása)*

Az alaptípusok halmaza most is legyen $K = \{Bool, Nat\}$. Az előző példa első részében már láttuk, hogy

$$x\!:\!Bool \to Nat\ \vdash wf,$$

és így az x típusára:

$$\frac{x\!:\!Bool \to Nat\ \vdash wf}{x\!:\!Bool \to Nat\ \vdash x\!:\!Bool \to Nat}$$

Most az xy applikáció típusát kell meghatároznunk, de ez csak akkor lehetséges, ha az y-nak is van egy levezethető jól formált típusa. Látható, hogy ennek a típusnak $Bool$-nak kell lennie, az y típusára azonban a kifejezésből semmilyen információt sem kapunk. Nincs más lehetőség, az xy típus meghatározásához az $y : Bool$ típust a Γ típuskörnyezetnek kell tartalmaznia, ezért a típuslevezetésben nem az $\emptyset$ üres típuskörnyezetből, hanem az $y : Bool$ jól formált típuskörnyezetből kell kiindulnunk. Tehát újrakezdve a levezetést:

$$\cfrac{\cfrac{\emptyset \vdash wf \quad Bool \in K}{\emptyset \vdash Bool} \qquad y \notin dom(\emptyset)}{y\!:\!Bool \vdash wf}$$

Ebből kiindulva, az példa elején levő levezetés:

$$\dfrac{\dfrac{\dfrac{y{:}Bool \vdash wf \quad Bool \in K}{y{:}Bool \vdash Bool} \qquad \dfrac{y{:}Bool \vdash wf \quad Nat \in K}{y{:}Bool \vdash Nat}}{y{:}Bool \vdash Bool \to Nat} \qquad x \notin dom(y{:}Bool)}{y{:}Bool,\ x{:}Bool \to Nat \vdash wf}$$

és így az x típusára:

$$\dfrac{y{:}Bool,\ x{:}Bool \to Nat \vdash wf}{y{:}Bool,\ x{:}Bool \to Nat \vdash x{:}Bool \to Nat}$$

és y-ra:

$$\dfrac{y{:}Bool,\ x{:}Bool \to Nat \vdash wf}{y{:}Bool,\ x{:}Bool \to Nat \vdash y{:}Bool}$$

Most az xy típusa már meghatározható:

$$\dfrac{\begin{array}{l} y{:}Bool,\ x{:}Bool \to Nat \vdash\ x{:}Bool \to Nat \\ y{:}Bool,\ x{:}Bool \to Nat \vdash\ y{:}Bool \end{array}}{y{:}Bool,\ x{:}Bool \to Nat \vdash xy{:}Nat}$$

A levezetést tovább folytatva, az absztrakció típusa:

$$\dfrac{y{:}Bool,\ x{:}Bool \to Nat \vdash xy{:}Nat}{y{:}Bool \vdash (\lambda x{:}Bool \to Nat\,.\,xy){:}(Bool \to Nat) \to Nat}$$

tehát az $\{y : Bool\}$ típuskörnyezetet feltételezve a $\lambda x{:}Bool \to Nat\,.\,xy$ kifejezés típusa $(Bool \to Nat) \to Nat$. □

3.3.6. Példa. *(Az Ω kifejezés típusa)*

Az F_1 rendszerben nem minden λ-kifejezésnek létezik típusa, a típusnélküli λ-kalkulusból megismert

$$\Omega \equiv (\lambda x\,.\,xx)(\lambda x\,.\,xx)$$

kifejezés például ilyen. Ezt a kifejezést az F_1 rendszerben felírva a következőt kapjuk:

$$\Omega \equiv (\lambda x{:}A\,.\,xx)(\lambda x{:}A\,.\,xx)$$

Az A típusnak függvénytípusnak kell lennie, hiszen az xx egy applikáció, és minden applikáció első tagja egy függvény, tehát A csak $B \to C$ formájú lehet, legyen tehát $A \equiv B \to C$. Azonban most az applikáció argumentuma is x, ezért az $A \equiv B$-nek is fenn kell állnia. Olyan B és C típusokat azonban nem találunk, amelyek kielégítik a $B \equiv B \to C$ egyenlőséget, azaz az Ω-hoz nem lehet típust hozzárendelni.

A típusnélküli λ-kalkulusban a $B = \ldots\ B\ \ldots$ egyenlet megoldására az Y

fixpont kombinátort használtuk úgy, hogy a jobboldal B-jére egy absztrakciót alkalmaztunk, és erre a B-vel, mint argumentummal egy applikációt adtunk meg. Ez a módszer azonban itt nem alkalmazható, mivel a B-re semmilyen „típusabsztrakciót” sem tudunk alkalmazni, mivel nincs ilyen fogalmunk sem. □

3.4. Az alaptípusok halmazának bővítése

Már utaltunk arra, hogy az elsőrendű típusos λ-kalkulusokban az alaptípusok halmazának megválasztása meghatározza a definiálható típusok halmazát. A példákban eddig a $\{Bool, Nat\}$ halmazt választottuk, most ezt a halmazt bővítjük, ugyanakkor az F_1 rendszert is kiegészítjük a *Bool* és *Nat* típusra és az új alaptípusokra vonatkozó szabályokkal. Ezzel az F_1 rendszer programnyelvi kapcsolatait is megmutatjuk.

3.4.1. A *Unit* típus

Programnyelvekben gyakran van szükség annak a jelölésére, hogy egy eljárásnak nincs argumentuma, vagy egy függvénynek nincs argumentuma és értéke. Ezt most a *Unit* típussal jelöljük, a típushoz egy érték tartozik, a unit. Megjegyezzük, hogy ezt a típust gyakran *Void*-nak vagy *Null*-nak is nevezik.

A *Unit* típusra nem adunk meg semmilyen műveletet, a típus szabályai csak azt állítják, hogy a *Unit* típus egy lehetséges típus, azaz jól formált, és azt, hogy a *Unit* típushoz csak egy érték, a unit tartozik.

$$\frac{\Gamma \vdash \mathit{wf}}{\Gamma \vdash \mathit{Unit}} \quad [\textsc{Type Unit}]$$

$$\frac{\Gamma \vdash \mathit{wf}}{\Gamma \vdash \mathsf{unit} : \mathit{Unit}} \quad [\textsc{Val Unit}]$$

3.4.2. A *Bool* típus

Az **??** pontban definiáltuk a true és false λ-kifejezéseket, ez a két érték alkotja a *Bool* típust. Ugyanitt láttuk az if λ-absztrakciót is, erre az utasításra most egy külön szabályt is meg tudunk adni.

$$\frac{\Gamma \vdash \mathit{wf}}{\Gamma \vdash \mathit{Bool}} \quad [\textsc{Type Bool}]$$

$$\frac{\Gamma \vdash wf}{\Gamma \vdash \mathsf{true}\!:\!Bool} \quad [\text{Val True}]$$

$$\frac{\Gamma \vdash wf}{\Gamma \vdash \mathsf{false}\!:\!Bool} \quad [\text{Val False}]$$

$$\frac{\Gamma \vdash E\!:\!Bool \quad \Gamma \vdash F\!:\!A \quad \Gamma \vdash G\!:\!A}{\Gamma \vdash (\mathsf{if}\ E\ F\ G)\!:\!A} \quad [\text{Fun If}]$$

3.4.3. A *Nat* típus

A számkonstansokat a **??** pontban a c_0 konstans és a succ függvény megadásával definiáltuk. Az így meghatározott értékek alkotják a *Nat* típust. A számjegyrendszerekre megadtuk még a pred és a zero függvényt is.

A *Nat* típusra a következő szabályokat adhatjuk meg:

$$\frac{\Gamma \vdash wf}{\Gamma \vdash Nat} \quad [\text{Type Nat}]$$

$$\frac{\Gamma \vdash wf}{\Gamma \vdash \mathsf{c}_0\!:\!Nat} \quad [\text{Val } c_0]$$

$$\frac{\Gamma \vdash E\!:\!Nat}{\Gamma \vdash \mathsf{succ}\ E\!:\!Nat} \quad [\text{Val Succ}]$$

$$\frac{\Gamma \vdash E\!:\!Nat}{\Gamma \vdash \mathsf{pred}\ E\!:\!Nat} \quad [\text{Fun Pred}]$$

$$\frac{\Gamma \vdash E\!:\!Nat}{\Gamma \vdash \mathsf{zero}\ E\!:\!Bool} \quad [\text{Fun Zero}]$$

3.4.4. A *Pair* típus

Adjuk meg a rendezett párra vonatkozó szabályokat. A rendezett párt adó pair függvényt és a párokra alkalmazható first és second függvényeket a **??** pontban definiáltuk. Ha a rendezett pár első eleme A_1, a második eleme A_2 típusú, akkor a párok lehetséges értékhalmazát, azaz a párok típusát jelöljük $Pair_{A_1 \times A_2}$-vel. Megjegyezzük, hogy ezt a típust ***szorzat típusnak*** is nevezzük, és $A_1 \otimes A_2$-vel, vagy egyszerűen $A_1 \times A_2$-vel is jelöljük.

A szabályok a következők:

$$\frac{\Gamma \vdash A_1 \quad \Gamma \vdash A_2}{\Gamma \vdash Pair_{A_1 \times A_2}} \quad [\text{Type Pair}_{A_1 \times A_2}]$$

$$\frac{\Gamma \vdash E_1 : A_1 \quad \Gamma \vdash E_2 : A_2}{\Gamma \vdash \mathsf{pair}\ E_1\ E_2 : Pair_{A_1 \times A_2}} \quad [\textsc{Val Pair}]$$

$$\frac{\Gamma \vdash E : Pair_{A_1 \times A_2}}{\Gamma \vdash \mathsf{first}\ E : A_1} \quad [\textsc{Fun First}]$$

$$\frac{\Gamma \vdash E : Pair_{A_1 \times A_2}}{\Gamma \vdash \mathsf{second}\ E : A_2} \quad [\textsc{Fun Second}]$$

3.4.5. A *Union* típus

A $Pair_{A_1 \times A_2}$ típus elemei olyan párok, amelyeknek első komponensei A_1, a második komponensei A_2 típusúak, azaz azt is mondhatjuk, hogy a pár típusa A_1 *és* A_2. Ha egy olyan típust definiálunk, amelynek elemei A_1 *vagy* A_2 típusúak, akkor a *Union* típust kapjuk meg, és a típust $Union_{A_1+A_2}$-vel jelöljük. Megjegyezzük, hogy ezt a típust gyakran ***diszjunkt összeg típusnak*** vagy röviden ***összeg típusnak*** nevezzük, és $A_1 \oplus A_2$-vel vagy $A_1 + A_2$-vel is jelöljük.

A *Union* típus az imperatív programnyelvekből jól ismert case utasítás használatára ad lehetőséget, ahol a típus egy elemének az aktuális típusa határozza meg, hogy a case utasítás melyik ágát kell végrehajtani.

A $Union_{A_1+A_2}$ típus konstruktora az $\mathsf{inLeft}_{A_1+A_2}$ és az $\mathsf{inRight}_{A_1+A_2}$ művelet. Azt, hogy egy $Union_{A_1+A_2}$ típusú kifejezés aktuális típusa A_1 vagy A_2, az isLeft és az isRight *Bool* értékű függvények határozzák meg. Ha a típus A_1, akkor az isLeft függvény true értéket, az isRight függvény false értéket ad, az A_2 típusra az értékek rendre false és true. A típustól függő elágazást a case művelet valósítja meg.

A szabályok a következők:

$$\frac{\Gamma \vdash A_1 \quad \Gamma \vdash A_2}{\Gamma \vdash Union_{A_1+A_2}} \quad [\textsc{Type Union}_{A_1+A_2}]$$

$$\frac{\Gamma \vdash E : A_1 \quad \Gamma \vdash A_2}{\Gamma \vdash \mathsf{inLeft}_{A_1+A_2}\ E : Union_{A_1+A_2}} \quad [\textsc{Val InLeft}]$$

$$\frac{\Gamma \vdash A_1 \quad \Gamma \vdash E : A_2}{\Gamma \vdash \mathsf{inRight}_{A_1+A_2}\ E : Union_{A_1+A_2}} \quad [\textsc{Val InRight}]$$

$$\frac{\Gamma \vdash E : Union_{A_1+A_2}}{\Gamma \vdash \mathsf{isLeft}\ E : Bool} \quad [\textsc{Fun IsLeft}]$$

$$\frac{\Gamma \vdash E : Union_{A_1+A_2}}{\Gamma \vdash \mathsf{isRight}\ E : Bool} \quad [\textsc{Fun IsRight}]$$

$$\frac{\Gamma \vdash E\colon \mathit{Union}_{A_1+A_2} \quad \Gamma \vdash F_1\colon B \quad \Gamma \vdash F_2\colon B}{\Gamma \vdash (\mathsf{case}\ E\ \mathsf{of}\ ((\mathsf{isLeft}\ E)\ \mathsf{then}\ F_1)\ \mathsf{or}\ ((\mathsf{isRight}\ E)\ \mathsf{then}\ F_2))\colon B}\ [\textsc{Fun Case}]$$

Az inLeft függvényt gyakran toLeft-nek, az inRight függvényt gyakran toRight-nak is nevezik.

A *Union* típussal a *Bool* típus is kifejezhető. Legyen

$\mathit{Bool} \equiv \mathit{Union}_{\mathit{Unit}+\mathit{Unit}}$.

Mivel a *Unit* típusnak csak egy eleme van, a *Union* típuskonstruktoraival csak két elem adható meg, ezt a két elemet nevezhetjük a true és false értékeknek.

$\mathsf{true} \equiv \mathsf{inLeft}_{\mathit{Unit}+\mathit{Unit}}\ \mathsf{unit}$,

$\mathsf{false} \equiv \mathsf{inRight}_{\mathit{Unit}+\mathit{Unit}}\ \mathsf{unit}$.

Ekkor a

$\mathsf{case}\ E\ \mathsf{of}\ ((\mathsf{isLeft}\ \mathsf{true})\ \mathsf{then}\ F_1)\ \mathsf{or}\ ((\mathsf{isRight}\ \mathsf{true})\ \mathsf{then}\ F_2)$

kifejezés $E \equiv$ true-ra az F_1, és hasonlóan, a false-ra az F_2 kifejezést adja, azaz a case egy *if then else* műveletnek felel meg.

3.4.6. A *Ref* típus

A típusrendszer lehetőséget ad arra is, hogy az imperatív nyelvekből jól ismert változó tartalmú memóriacellákkal is foglalkozzunk. Ha A egy típus, akkor jelöljük *Ref* A-val az A típusú értékeket tartalmazó cellákat, azaz a *Ref* A típushalmaz elemei legyenek ezek a memóriacellák. Egy új cellát az alloc művelettel lehet allokálni, a dealloc művelettel megszüntetni, és egy cella értékét az értértékadást jelző $:=$ művelettel lehet megváltoztatni. Mivel az értékadás egy *mellékhatást* eredményez, az értékadás típusa legyen *Unit*.

$$\frac{\Gamma \vdash A}{\Gamma \vdash \mathit{Ref}\ A}\ [\textsc{Type Ref}]$$

$$\frac{\Gamma \vdash M\colon A}{\Gamma \vdash \mathsf{alloc}\ M\colon \mathit{Ref}\ A}\ [\textsc{Val Alloc}]$$

$$\frac{\Gamma \vdash M\colon \mathit{Ref}\ A}{\Gamma \vdash \mathsf{dealloc}\ M\colon A}\ [\textsc{Val Dealloc}]$$

$$\frac{\Gamma \vdash M\colon \mathit{Ref}\ A \quad \Gamma \vdash N\colon A}{\Gamma \vdash M := N\colon \mathit{Unit}}\ [\textsc{Fun Assign}]$$

3.4.7. A *List* típus

A **??** pontban vizsgáltuk a láncolt lista adatszerkezeteket, és megadtuk a nil és cons konstruktorok, a head, tail és empty függvények λ-kifejezéseit.

Az A típusú elemeket tartalmazó lista típusát $List_A$-val jelöljük, erre a típusra a következő szabályokat adhatjuk meg:

$$\frac{\Gamma \vdash A}{\Gamma \vdash List_A} \; [\textsc{Type List}_A]$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash \mathsf{nil}\!:\!List_A} \; [\textsc{Val Nil}]$$

$$\frac{\Gamma \vdash E\!:\!A \quad \Gamma \vdash F\!:\!List_A}{\Gamma \vdash \mathsf{cons}\ E\ F\!:\!List_A} \; [\textsc{Val Cons}]$$

$$\frac{\Gamma \vdash E\!:\!List_A}{\Gamma \vdash \mathsf{head}\ E\!:\!A} \; [\textsc{Fun Head}]$$

$$\frac{\Gamma \vdash E\!:\!List_A}{\Gamma \vdash \mathsf{tail}\ E\!:\!List_A} \; [\textsc{Fun Tail}]$$

$$\frac{\Gamma \vdash E\!:\!List_A}{\Gamma \vdash \mathsf{empty}\ E\!:\!Bool} \; [\textsc{Fun Empty}]$$

Látható, hogy a nil konstanshoz is hozzá kell rendelni a $List_A$ típust, azért, hogy a lista utolsó, a nil-t tartalmazó eleme is jól típusozott legyen.

3.4.8. A *Record* típus

A *Record* típus a *Pair* rendezett pár típus olyan általánosításának tekinthető, ahol a párt n-esre bővítjük, és az n-es minden eleméhez egy-egy címkét rendelünk.

Egy rekord mezőkből áll, a mezőknek neveik vannak, minden mező egy megadott típusú adatot tartalmaz.

Egy n-elemű rekord mezőnevei, azaz a mezők címkéi legyenek l_i-k ($1 \leq i \leq n$), az l_i mező adatának a típusa pedig legyen A_i. Ekkor a rekord típusa

$Record\{l_1\!:\!A_1, \ldots, l_n\!:\!A_n\}$,

vagy röviden

$\{\!| l_1\!:\!A_1, \ldots, l_n\!:\!A_n |\!\}$.

Ha az $E\!:\!\{\!| l_1\!:\!A_1, \ldots, l_n\!:\!A_n |\!\}$ rekord mezőit az $F_1\!:\!A_1, \ldots, F_n\!:\!A_n$ kifejezésekkel fel akarjuk tölteni, akkor ezt a $\mathsf{record}\ l_1 = F_1\ \ldots\ l_n = F_n$ *konstruktorral* tehetjük meg. Ennek a rekordnak a rövid leírására a $\{l_1 = F_1 \ \ldots \ l_n = F_n\}$ jelölést is használni fogjuk.

Egy rekordhoz definiálhatunk selector l_i $(1 \leq i \leq n)$ függvényeket is, amelyek a rekord l_i nevű mezőjének adatát határozzák meg:

selector l_i (record $l_1 = F_1 \ \ldots \ l_n = F_n$) $\rightarrow$ F_i.

A selector l_i E applikációt gyakran $E\,.\,l_i$-vel, vagy inkább az E^l_i jellel is jelölik.

A *Record* típusra vonatkozó szabályok a következők:

$$\frac{\Gamma \vdash A_1 \ \ldots \ \Gamma \vdash A_n}{\Gamma \vdash \{\!|l_1 {:} A_1, \ldots, l_n {:} A_n |\!\}} \quad [\textsc{Type Record}]$$

$$\frac{\Gamma \vdash F_1 {:} A_1 \ \ldots \ \Gamma \vdash F_n {:} A_n}{\Gamma \vdash \mathsf{record}\ l_1 = F_1 \ \ldots \ l_n = F_n {:} \{\!|l_1 {:} A_1, \ldots, l_n {:} A_n |\!\}} \quad [\textsc{Val Record}]$$

$$\frac{\Gamma \vdash \mathsf{record}\ l_1 = F_1 \ \ldots \ l_n = F_n {:} \{\!|l_1 {:} A_1, \ldots, l_n {:} A_n |\!\}}{\Gamma \vdash \mathsf{selector}\ l_i\ (\mathsf{record}\ l_1 = F_1 \ \ldots \ l_n = F_n) {:} A_i} \quad [\textsc{Fun Selector}]$$

3.4.9. A *Variant* típus

Míg a *Record* típus a *Pair*, a *Variant* típus a két komponensű *Union* típus általánosítása. A *Union* típust több komponensre bővítjük, és minden komponenshez egy címkét rendelünk.

Egy n komponensű *Variant* típus komponenseinek címkéi legyenek l_i-k $(1 \leq i \leq n)$, az l_i komponens típusa legyen A_i. Ekkor a *Variant* típus így írható le:

$\mathit{Variant}\{l_1 {:} A_1, \ldots, l_n {:} A_n\}$.

A típus konstruktorát nevezzük inVariant-nak, és az aktuális típus vizsgálatát végezze egy isVariant függvény. Az erre a típusra alkalmazható case többirányú elágazást hajt végre.

A *Variant* típusra vonatkozó szabályok a következők:

$$\frac{\Gamma \vdash A_1 \ \ldots \ \Gamma \vdash A_n}{\Gamma \vdash \mathit{Variant}\{l_1 {:} A_1, \ldots, l_n {:} A_n\}} \quad [\textsc{Type Variant}_{A_1, A_2, \ldots, A_n}]$$

$$\frac{\Gamma \vdash A_1 \ \ldots \ \Gamma \vdash A_n \quad \Gamma \vdash E {:} A_i}{\Gamma \vdash \mathsf{inVariant}_{\mathit{Variant}\{l_1 {:} A_1, \ldots, l_n {:} A_n\}}\ l_i = E {:} \mathit{Variant}\{l_1 {:} A_1, \ldots, l_n {:} A_n\}}$$

[Val InVariant]

$$\frac{\Gamma \vdash E {:} \mathit{Variant}\{l_1 {:} A_1, \ldots, l_n {:} A_n\}}{\Gamma \vdash \mathsf{isVariant}\ l_i\ E {:} \mathit{Bool}} \quad [\textsc{Fun IsVariant}]$$

$$\frac{\Gamma \vdash E\colon \mathit{Variant}\{l_1\!:\!A_1,\ldots,l_n\!:\!A_n\} \quad \Gamma \vdash F_1\!:\!B \ \ldots\ \Gamma \vdash F_n\!:\!B}{\Gamma \vdash (\mathsf{case}\, E\, \mathsf{of}((\mathsf{isVariant}\, l_1\, E)\mathsf{then}\, F_1)\mathsf{or}\ldots\mathsf{or}((\mathsf{isVariant}\, l_n\, E)\mathsf{then}\, F_n)) : B}$$

[FUN CASE]

3.5. Az alaptípusok definiálása

A 3.2.1. definícióban láttuk, hogy az F_1 rendszer típusai az *alaptípusok* halmazának elemeiből épülnek fel, az alaptípusok halmaza egy adott F_1 rendszerben nem változtatható meg.

Most egy olyan módszert mutatunk be, amellyel a predefinit alaptípusok explicit felsorolását elhagyhatjuk, és az alaptípusokat az *induktív típusdefiníciónak* nevezett művelettel definiálhatjuk, a típus elemeit pedig a definícióban levő konstruktorokkal állíthatjuk elő. Az ezzel a definícióval megadott alaptípusú F_1 rendszert F_1^{ind} rendszernek nevezzük, ahol az *ind* az induktív szóra utal.

Az induktív típusdefiníciót az indtype kulcsszó vezeti be, utána található a definiálandó új típus neve, majd a with kulcsszó után a *konstruktor* : *típus* párok felsorolása.

3.5.1. Definíció. Induktív típusdefiníció:

Az F_1^{ind} rendszer szintaktikája csak abban különbözik az F_1 rendszer 3.2.1. definícióban megadott szintaktikájától, hogy a < típus > alaptípus megadását most az indtype *művelettel végezzük.*

$$\begin{array}{llll} \mathsf{indtype}\ <\mathit{típus}>\ \mathsf{with} & E_1 & : & T_{11} \to T_{12} \to \ldots \to T_{1m_1} \to <\mathit{típus}> \\ & E_2 & : & T_{21} \to T_{22} \to \ldots \to T_{2m_2} \to <\mathit{típus}> \\ & \ldots & & \\ & E_n & : & T_{n1} \to T_{n2} \to \ldots \to T_{nm_n} \to <\mathit{típus}> \end{array}$$

ahol

- E_i $(1 \le i \le n)$ *a típus típuskonstruktora, speciálisan lehet az új típus értékhalmazának egy eleme, azaz a típus konstansa is. A λ-kifejezések szintaktikájának leírását a konstruktorokkal bővíteni kell, azaz*

  $<\lambda\text{-}\mathit{kifejezés}> = \ \ldots\ |\ E_1\ |\ E_2\ |\ \ldots\ |\ E_n,$

- T_{ij} $(1 \le i \le n, 1 \le j \le m_i)$ *az éppen ezzel a típusdefinícióval megadott < típus > típusból és a már korábban megadott, ismert típusokból készített jól-formált típus.*

A definícióból látható, hogy ha E_i-t applikáljuk például m_i darab, rendre $T_{i1}, T_{i2}, \ldots, T_{im_i}$ típusú argumentumra, akkor egy indtype-pal definiált <

típus > típusú eredményt kapunk.

Látható az is, hogy ez a definíció konstansos λ-kifejezéseket ad, ahol a konstruktorok a λ-kalkulus δ-függvényeinek felelnek meg, a speciális konstans típusértékek pedig a λ-kalkulus konstansai lesznek.

3.5.2. Példa. *(A konstruktor nélküli típus)*
Az induktív típusdefiníció lehetőséget ad olyan típus definiálására is, amelynek egyáltalán nincs konstruktora, ami azt jelenti, hogy nincs egyetlen olyan λ-kifejezés sem, amelynek ez lenne a típusa. Nevezzük ezt a típust *Absurd*-nak, így a típus definíciója

$$\mathsf{indtype}\ Absurd\ \mathsf{with}\ \emptyset \qquad \square.$$

3.5.3. Példa. *(A Unit típus)*
A *Unit* típus értékhalmaza egy elemet tartalmaz, ez a típus a következőképpen adható meg:

$$\mathsf{indtype}\ Unit\ \mathsf{with}\ \mathsf{unit} : Unit$$

Nyilvánvaló, hogy a típus unit konstruktora konstans. □

3.5.4. Példa. *(A Bool típus)*
A logikai, azaz a *Bool* típus megadása induktív típusdefinícióval a következő:

$$\begin{array}{lrl} \mathsf{indtype}\ Bool\ \mathsf{with} & \mathsf{true} : & Bool \\ & \mathsf{false} : & Bool \end{array}$$

Látható, hogy a *Bool* típus mindkét konstruktora konstans. □

3.5.5. Példa. *(A Nat típus induktív típusdefinícióval)*
A természetes számok *Nat* típusa a következőképpen írható le: a következő:

$$\begin{array}{lrl} \mathsf{indtype}\ Nat\ \mathsf{with} & 0 : & Nat \\ & \mathsf{succ} : & Nat \to Nat \end{array} \qquad \square$$

Most megmutatjuk, hogy a fenti példában definiált *Nat* típus értékhalmazának elemei hogyan határozhatók meg. Az értékhalmaz megadására az iter *primitív rekurzív függvényt* használjuk:

3.5.6. Definíció. Az iter függvény:

$$\begin{array}{llll} \mathsf{iter} & 0 & x\ f & \twoheadrightarrow\ x \\ \mathsf{iter} & (\mathsf{succ}\ y) & x\ f & \twoheadrightarrow\ f\ (\mathsf{iter}\ y\ x\ f) \end{array}$$

A $\twoheadrightarrow$ jel a redukciót jelzi. Az iter függvény szemléletesen azt jelenti, hogy az f-t az x-re pontosan az első argumentumban megadott darabszámszor alkalmazzuk.

Az iter függvény az első argumentumban megadott tetszőleges számkonstansra biztosan termináló redukció-sorozatot ad, az iter függvény bevezetése a λ-kifejezések erős normalizálását nem változtatja meg.

Az n természetes szám a következő kifejezéssel adható meg:

$\mathsf{iter}\ n\ 0\ \mathsf{succ}$,

és ebből a Nat típusú tetszőleges n számot előállító függvény:

$\lambda x : Nat.\ \mathsf{iter}\ x\ 0\ \mathsf{succ}$.

3.5.7. Példa. *(Az iter függvény felhasználása)*

Megjegyezzük, hogy a Nat típusú számok megadásán kívül az iter függvényt más célra is fel lehet használni, például az $E_{2x+1}\ n = 2n + 1$, $E_{x+y}\ a\ b = a + b$ és az $E_{x*y}\ a\ b = a * b$ műveletek a következő kifejezésekkel adhatók meg:

$$E_{2x+1} \equiv \lambda x : Nat.\ \mathsf{iter}\ x\ (\mathsf{succ}\ 0)\ (\lambda z : Nat.\ \mathsf{succ}\ (\mathsf{succ}\ z)),$$
$$E_{x+y} \equiv \lambda x : Nat.\ \lambda y : Nat.\ \mathsf{iter}\ x\ y\ \mathsf{succ},$$
$$E_{x*y} \equiv \lambda x : Nat.\ \lambda y : Nat.\ \mathsf{iter}\ x\ 0\ (\lambda z : Nat.\ E_{z+y}\ z\ y).$$

Mindhárom kifejezés típusa $Nat \to Nat \to Nat$. □

Az iter függvény első argumentuma és a függvény értéke csak Nat típusú lehet. Látható, hogy az iter függvény a Nat típus elemein struktúrális indukciót hajt végre, és az indukció eredménye Nat típusú.

3.5.8. Példa. *(A $Pair_{A_1 \times A_2}$ és a $List_A$ típus induktív típusdefinícióval)*

$\mathsf{indtype}\ Pair_{A_1 \times A_2}\ \mathsf{with}\ \ \mathsf{pair} : A_1 \to A_2 \to Pair_{A_1 \times A_2}$

$\mathsf{indtype}\ List_A\ \mathsf{with}\ \mathsf{nil} \quad : List_A$

$\qquad\qquad\qquad\qquad \mathsf{cons} : A \to List_A \to List_A$ □

Általánosítsuk az iter függvényt úgy, hogy a struktúrális indukciót egy tetszőleges A típus elemeire hajtsa végre, és eredményül egy B típusú értéket adjon. Ezt a függvényt jelöljük iter_A_B-vel. Ezzel a jelöléssel az előbbi iter függvény neve most iter_Nat_Nat lesz.

3.5.9. Definíció. Az iter_A_B függvény:

Ha az A típust megadó induktív típusdefinícióban n konstruktor van, és az i-edik $(1 \leq i \leq n)$ konstruktor

$E_i : T_{i1} \to T_{i2} \to \ldots \to T_{im_i} \to A$

alakú, akkor

$\mathsf{iter\_A\_B}\ (E_i\ G_1 G_2 \ldots G_{m_i})\ F_1 F_2 \ldots F_n \quad \twoheadrightarrow \quad F_i\ G'_1 G'_2 \ldots G'_{m_i},$

ahol

- *F_i egy $T'_{i1} \to T'_{i2} \to \ldots \to T'_{im_i} \to B$ típusú függvény,*
- *G'_j azt jelöli, hogy G_j-ben minden A típusú t részkifejezést $(\mathsf{iter\_A\_B}\ t\ F_1 F_2 \ldots F_n)$-nel helyettesítünk,*
- *és T'_{ik} jelöli azt, hogy T_{ik}-ban az A minden előfordulását B-re helyettesítjük.*

Ha az E_i konstruktornak nincs paramétere, akkor a fentiek szerint

$\mathsf{iter\_A\_B}\ E_i\ F_1\ F_2\ \ldots\ F_n \quad \twoheadrightarrow \quad F_i,$

és ha az E_i konstruktor egyik paraméterének sincs A típusú részkifejezése, akkor

$\mathsf{iter\_A\_B}\ (E_i\ a_1 a_2 \ldots\ a_{m_i})\ F_1 F_2 \ldots F_n \quad \twoheadrightarrow \quad F_i\ a_1 a_2 \ldots a_{m_i}.$

3.5.10. Példa. *(Az* iter_Bool_Bool *függvény)*
Mint láttuk, a *Bool* típus induktív típusdefiníciója:

$$\begin{array}{lll} \mathsf{indtype}\ Bool\ \mathsf{with} & \mathsf{true} & : Bool \\ & \mathsf{false} & : Bool \end{array}$$

így, mivel true az első és false a második konstruktor,

$\mathsf{iter\_Bool\_Bool\ true}\ F_1 F_2 \ \twoheadrightarrow\ F_1$

$\mathsf{iter\_Bool\_Bool\ false}\ F_1 F_2 \ \twoheadrightarrow\ F_2$ □

3.5.11. Példa. *(Az iter_Bool_Bool alkalmazása)*
Az and, or és not függvények leírására az iter_Bool_Bool függvényt használhatjuk:

$\mathsf{and} \equiv \lambda x : Bool.\,\lambda y : Bool.\,\mathsf{iter\_Bool\_Bool}\ xy\ \mathsf{false},$

$\mathsf{or} \ \ \equiv \lambda x : Bool.\,\lambda y : Bool.\,\mathsf{iter\_Bool\_Bool}\ x\ \mathsf{true}\ y,$

$\mathsf{not} \equiv \lambda x : Bool.\,\mathsf{iter\_Bool\_Bool}\ x\ \mathsf{false\ true}.$

Például,

$\mathsf{not\ true} \twoheadrightarrow_\beta \mathsf{iter\_Bool\_Bool\ true\ false\ true} \twoheadrightarrow \mathsf{false}$ □

3.5.12. Példa. *(Az* iter_Bool_Nat *függvény)*
Az iter_Bool_Bool függvényhez hasonlóan,

$\mathsf{iter\_Bool\_Nat\ true}\ F_1 F_2 \twoheadrightarrow F_1$

$\mathsf{iter\_Bool\_Nat\ false}\ F_1 F_2 \twoheadrightarrow F_2$ □

3.5.13. Példa. *(Az iter_Bool_Nat alkalmazása)*
Az *if x : Bool then y : Nat else z : Nat* szerkezet leírására az iter_Bool_Nat függvényt használhatjuk:

$\mathsf{if} \equiv \lambda x : Bool\,.\,\lambda y : Nat\,.\,\lambda z : Nat\,.\,\mathsf{iter\_Bool\_Nat}\ xyz$

Például, ha y és z értéke 2 és 3, azaz *Nat* típusú, akkor

$\mathsf{if\ false}\ 2\ 3 \twoheadrightarrow_\beta \mathsf{iter\_Bool\_Nat\ false}\ 2\ 3 \twoheadrightarrow 3,$

a *Bool* definíciójában a false a második konstruktor, ezért a iter_Bool_Nat-nak a *Bool* típusú argumentum utáni második argumentumát kapjuk eredményül. □

3.5.14. Példa. *(A $Pair_{A_1 \times A_2}$ típus)*
Nat típusú számokból képezzük a $Pair_{Nat\times Nat}$ típust. A típus leírása induktív típusdefinícióval a következő:

$\mathsf{indtype}\ Pair_{Nat\times Nat}\ \mathsf{with\ pair}{:}Nat \to Nat \to Pair_{Nat\times Nat}$

Egy rendezett pár első és második elemét az $\mathsf{iter\_Pair_{Nat\times Nat}\_Nat}$ függvénnyel adhatjuk meg:

$\mathsf{first} \equiv \lambda x{:}Pair_{Nat\times Nat}\,.\,\mathsf{iter\_Pair_{Nat\times Nat}\_Nat}\ x\ (\lambda f{:}Nat\,.\,\lambda s{:}Nat\,.\,f)$

$\mathsf{second} \equiv \lambda x{:}Pair_{Nat\times Nat}\,.\,\mathsf{iter\_Pair_{Nat\times Nat}\_Nat}\ x\ (\lambda f{:}Nat\,.\,\lambda s{:}Nat\,.\,s)$

Mivel a $Pair_{Nat\times Nat}$ típusnak egy konstruktora van, a pár első vagy második elemét az $\mathsf{iter\_Pair_{Nat\times Nat}\_Nat}$ függvény második argumentumában levő kifejezés választja ki. Míg a pair konstruktor típusa $Nat \to Nat \to Pair_{Nat\times Nat}$, ennek a kifejezésnek a típusa $Nat \to Nat \to Nat$.

Például a pair uv pár első eleme:

$\mathsf{first}\ (\mathsf{pair}\ uv) \equiv$

$(\lambda x{:}Pair_{Nat\times Nat}\,.\,\mathsf{iter\_Pair_{Nat\times Nat}\_Nat}\ x\ (\lambda f{:}Nat\,.\,\lambda s{:}Nat\,.\,f))\ (\mathsf{pair}\ uv) \twoheadrightarrow_\beta$

$\mathsf{iter\_Pair_{Nat\times Nat}\_Nat}\ (\mathsf{pair}\ uv)\ (\lambda f{:}Nat\,.\,\lambda s{:}Nat\,.\,f) \twoheadrightarrow$

$(\lambda f{:}Nat\,.\,\lambda s{:}Nat\,.\,f)\ uv \twoheadrightarrow_\beta$

$(\lambda s{:}Nat\,.\,u)\ v \twoheadrightarrow_\beta u$ □

3.5.15. Példa. *(A láncolt lista adatszerkezet List típusa)*

A Nat típusú számokat tartalmazó láncolt lista megadása a következő:

$$\begin{aligned}\mathsf{indtype}\ List\ \mathsf{with}\ \mathsf{nil}\ &: List\\ \mathsf{cons}&: Nat\ \to\ List\ \to\ List\end{aligned}$$

A lista fejelemét egy iter_List_Nat függvénnyel határozzuk meg, mivel a fejelem típusa Nat. A problémát azonban az okozza, hogy így egy Nat típusú értéket kell adnunk akkor is, ha a lista üres. Ezért válasszunk ki egy számot, ez a szám legyen az üres lista „jele", azaz a hibajelzés, és ez a szám legyen a head második paramétere. Így

$$\mathsf{head} \equiv \lambda x{:}List\,.\,\lambda d{:}Nat\ .\,\mathsf{iter\_List\_Nat}\ xd(\lambda c{:}Nat\,.\,\lambda r{:}List\,.\,c)$$

Határozzuk meg a $[1 :: 2 :: 3 :: 4] \equiv \mathsf{cons}\ 1(\mathsf{cons}\ 2(\mathsf{cons}\ 3(\mathsf{cons}\ 4\ \mathsf{nil})))$ lista fejelemét. A hibajelzés kódja legyen 999.

$$\mathsf{head}\ (\mathsf{cons}\ 1(\mathsf{cons}\ 2(\mathsf{cons}\ 3(\mathsf{cons}\ 4\ \mathsf{nil}))))\ 999 \equiv$$

$$\lambda x{:}List\,.\,\lambda d{:}Nat\ .\,\mathsf{iter\_List\_Nat}\ xd\ (\lambda c{:}Nat\,.\,\lambda r{:}List\,.\,c)$$
$$(\mathsf{cons}\ 1(\mathsf{cons}\ 2(\mathsf{cons}\ 3(\mathsf{cons}\ 4\ \mathsf{nil}))))\ 999\ \twoheadrightarrow_\beta^+$$

$$\mathsf{iter\_List\_Nat}\ (\mathsf{cons}\ 1(\mathsf{cons}\ 2(\mathsf{cons}\ 3(\mathsf{cons}\ 4\ \mathsf{nil}))))\ 999\ (\lambda c{:}Nat\,.\,\lambda r{:}List\,.\,c)\ \twoheadrightarrow$$

$$(\lambda c{:}Nat\,.\,\lambda r{:}List\,.\,c)\ 1\ \underline{(\mathsf{cons}\ 2\ (\mathsf{cons}\ 3(\mathsf{cons}\ 4\ \mathsf{nil})))}$$

Mivel az aláhúzott kifejezés $List$ típusú, az iter_List_Nat redukciójának végrehajtásához ezt a kifejezést tovább kell alakítani:

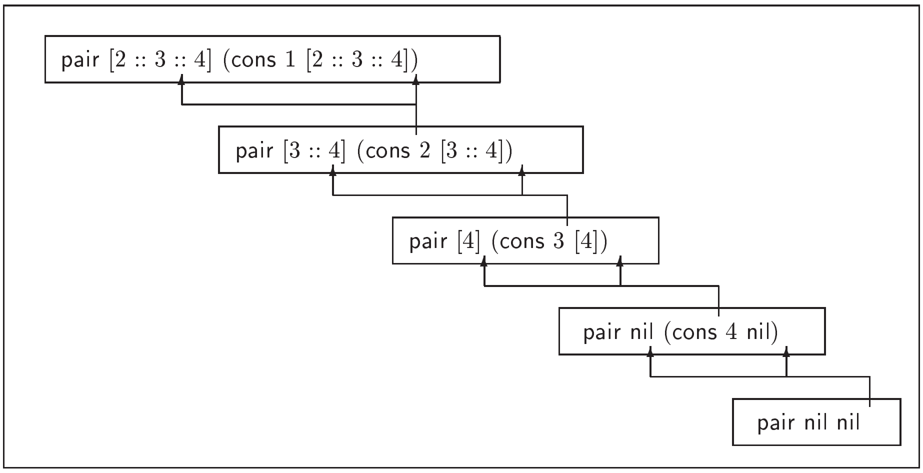
$$\mathsf{cons}\ 2(\mathsf{cons}\ 3(\mathsf{cons}\ 4\ \mathsf{nil}))\ \twoheadrightarrow$$

$$\mathsf{iter\_List\_Nat}\ (\mathsf{cons}\ 2(\mathsf{cons}\ 3(\mathsf{cons}\ 4\ \mathsf{nil})))\ 999\ (\lambda c{:}Nat\,.\,\lambda r{:}List\,.\,c)\ \twoheadrightarrow$$

$$(\lambda c{:}Nat\,.\,\lambda r{:}List\,.\,c)\ 2\ \underline{(\mathsf{cons}\ 3(\mathsf{cons}\ 4\ \mathsf{nil}))}$$

Az aláhúzott kifejezés átalakítását az olvasóra bízzuk. Látható, hogy ezek az átalakítások az eredményt nem befolyásolják, hiszen a $\lambda c{:}Nat\,.\,\lambda r{:}List\,.\,c$ függvény eredményül az első argumentumot adja, és így eredményül valóban a lista $1{:}Nat$ elemét kapjuk meg.

Egy lista maradékrészének megadása kissé bonyolult. A maradék részt úgy a legegyszerűbb felépíteni, hogy a lista építését jobboldalról, a lista utolsó elemétől kezdjük, és mire a lista első eleméhez érünk, az ekkor alkalmazott cons konstruktor második argumentuma éppen a meghatározandó maradék rész. Azonban ehhez nem elég csak a maradék részeket, azaz a cons konstruktor második argumentumait nyilvántartani, hiszen egy cons konstruktor második argumentumát az előző lépésben létrehozott teljes lista alkotja. Erre a problémára a szokásos megoldás az, hogy párokat hozunk létre, egy pár első eleme a lista maradék részét, a második eleme pedig a teljes listát tartalmazza. Így a lista felépítése után az eredményül kapott páros első eleme éppen a lista maradék részét tartalmazza (3.1. ábra).



3.1. ábra. Az [1 :: 2 :: 3 :: 4] lista

Először adjuk meg a listákat tartalmazó párok típusát:

$\mathsf{indtype}\ Pair_{List\times List}\ \mathsf{with}\ \mathsf{pair}{:}List\rightarrow List\rightarrow Pair_{List\times List}$

Ekkor a maradék rész a következő.

$\mathsf{tail}\equiv\lambda l{:}List\,.\,\mathsf{first}(\mathsf{iter\_List\_Pair}_{List\times List}\ l(\mathsf{pair\ nil\ nil})$
$\qquad(\lambda c{:}Nat\,.\,\lambda r{:}Pair_{List\times List}\,.\,\mathsf{pair}(\mathsf{second}\ r)(\mathsf{cons}\ c(\mathsf{second}\ r)))$

Határozzuk meg az üres lista maradék részét.

$\mathsf{tail\ nil}\equiv$
$(\lambda l{:}List\,.\,\mathsf{first}(\mathsf{iter\_List\_Pair}_{List\times List}\ l(\mathsf{pair\ nil\ nil})$
$\qquad(\lambda c{:}Nat\,.\,\lambda r{:}Pair_{List\times List}\,.\,\mathsf{pair}(\mathsf{second}\ r)(\mathsf{cons}\ c(\mathsf{second}\ r)))))\mathsf{nil}\ \twoheadrightarrow_\beta$
$\mathsf{first}(\mathsf{iter\_List\_Pair}_{List\times List}\ \mathsf{nil}(\mathsf{pair\ nil\ nil})$
$\qquad(\lambda c{:}Nat\,.\,\lambda r{:}Pair_{List\times List}\,.\,\mathsf{pair}(\mathsf{second}\ r)(\mathsf{cons}\ c(\mathsf{second}\ r))\ \twoheadrightarrow$
$\mathsf{first}(\mathsf{pair\ nil\ nil})\ \twoheadrightarrow^+_\beta$
nil

Határozzuk meg az $[1 :: 2 :: 3 :: 4] \equiv \mathsf{cons}\ 1(\mathsf{cons}\ 2(\mathsf{cons}\ 3(\mathsf{cons}\ 4\ \mathsf{nil})))$ lista maradék részét.

$\mathsf{tail}(\mathsf{cons}\ 1(\mathsf{cons}\ 2(\mathsf{cons}\ 3(\mathsf{cons}\ 4\ \mathsf{nil})))) \equiv$

$(\lambda l{:}List\,.\,\mathsf{first}(\mathsf{iter\_List\_Pair}_{List\times List}\ l(\mathsf{pair\ nil\ nil})$
$\quad(\lambda c{:}Nat\,.\,\lambda r{:}Pair_{List\times List}\,.\,\mathsf{pair}(\mathsf{second}\ r)(\mathsf{cons}\ c(\mathsf{second}\ r)))))$
$\quad(\mathsf{cons}\ 1(\mathsf{cons}\ 2(\mathsf{cons}\ 3(\mathsf{cons}\ 4\ \mathsf{nil})))) \twoheadrightarrow_\beta$

$\mathsf{first}(\mathsf{iter\_List\_Pair}_{List\times List}\ (\mathsf{cons}\ 1(\mathsf{cons}\ 2(\mathsf{cons}\ 3(\mathsf{cons}\ 4\ \mathsf{nil}))))(\mathsf{pair\ nil\ nil})$
$\quad(\lambda c{:}Nat\,.\,\lambda r{:}Pair_{List\times List}\,.\,\mathsf{pair}(\mathsf{second}\ r)(\mathsf{cons}\ c(\mathsf{second}\ r)))) \twoheadrightarrow$

$\mathsf{first}(\lambda c{:}Nat\,.\,\lambda r{:}Pair_{List\times List}\,.\,\mathsf{pair}(\mathsf{second}\ r)(\mathsf{cons}\ c(\mathsf{second}\ r))$
$\quad 1\ \underline{(\mathsf{cons}\ 2(\mathsf{cons}\ 3(\mathsf{cons}\ 4\ \mathsf{nil}))}$

A fejelem meghatározásánál látottakhoz hasonlóan, mivel az az aláhúzott kifejezés *List* típusú, az $\mathsf{iter\_List\_Pair}_{List\times List}$ redukciójának végrehajtásához ezt a kifejezést tovább kell alakítani:

$(\mathsf{iter\_List\_Pair}_{List\times List}\ (\mathsf{cons}\ 2(\mathsf{cons}\ 3(\mathsf{cons}\ 4\ \mathsf{nil})))(\mathsf{pair\ nil\ nil})$
$\quad(\lambda c{:}Nat\,.\,\lambda r{:}Pair_{List\times List}\,.\,\mathsf{pair}(\mathsf{second}\ r)(\mathsf{cons}\ c(\mathsf{second}\ r)))) \twoheadrightarrow$

$(\lambda c{:}Nat\,.\,\lambda r{:}Pair_{List\times List}\,.\,\mathsf{pair}(\mathsf{second}\ r)(\mathsf{cons}\ c(\mathsf{second}\ r)))$
$\quad 2\ \underline{(\mathsf{cons}\ 3(\mathsf{cons}\ 4\ \mathsf{nil}))}$

A redukciót tovább folytatva

$(\mathsf{iter\_List\_Pair}_{List\times List}\ (\mathsf{cons}\ 3(\mathsf{cons}\ 4\ \mathsf{nil}))(\mathsf{pair\ nil\ nil})$
$\quad(\lambda c{:}Nat\,.\,\lambda r{:}Pair_{List\times List}\,.\,\mathsf{pair}(\mathsf{second}\ r)(\mathsf{cons}\ c(\mathsf{second}\ r)))) \twoheadrightarrow$

$(\lambda c{:}Nat\,.\,\lambda r{:}Pair_{List\times List}\,.\,\mathsf{pair}(\mathsf{second}\ r)(\mathsf{cons}\ c(\mathsf{second}\ r)))$
$\quad 3\ \underline{(\mathsf{cons}\ 4\ \mathsf{nil})}$

majd

$(\mathsf{iter\_List\_Pair}_{List\times List}\ (\mathsf{cons}\ 4\ \mathsf{nil})(\mathsf{pair\ nil\ nil})$
$\quad(\lambda c{:}Nat\,.\,\lambda r{:}Pair_{List\times List}\,.\,\mathsf{pair}(\mathsf{second}\ r)(\mathsf{cons}\ c(\mathsf{second}\ r)))) \twoheadrightarrow$

$(\lambda c{:}Nat\,.\,\lambda r{:}Pair_{List\times List}\,.\,\mathsf{pair}(\mathsf{second}\ r)(\mathsf{cons}\ c(\mathsf{second}\ r)))$
$\quad 4\ \underline{\mathsf{nil}}$

és végül

$(\mathsf{iter\_List\_Pair}_{List\times List}\ (\mathsf{nil})(\mathsf{pair\ nil\ nil})$
$\quad(\lambda c{:}Nat\,.\,\lambda r{:}Pair_{List\times List}\,.\,\mathsf{pair}(\mathsf{second}\ r)(\mathsf{cons}\ c(\mathsf{second}\ r)))) \twoheadrightarrow$

$\mathsf{pair\ nil\ nil}$

azaz a 3.1. ábra utolsó sorában szereplő kifejezést kaptuk meg. Ha ezt a nil kifejezés helyére írjuk és végrehajtjuk a kifejezésben megadott β-redukciókat, akkor éppen az ábra utolsó előtti sorának kifejezését kapjuk. Végrehajtva a iter_List_Pair$_{List \times List}$ redukció minden lépését, látható, hogy a redukció éppen az ábra első sorának kifejezését határozza meg, és ennek a párosnak az első tagja a lista keresett tail része. □

.

3.6. Az F_1 rendszer operációs szemantikája

Először a típuskörnyezetekre adunk meg néhány egyszerű állítást. A 4.1.2. definíció szerint a típuskörnyezet $x_i : A_i$ $(1 \leq i \leq n)$ párok listája. Az első állítás azt mondja ki, hogy a jól típusozottságot a lista elemeinek sorrendje nem befolyásolja. A következő két lemmában a típuskörnyezetet módosítjuk. A második állítás azt mondja ki, hogy a típuskörnyezet bővítése a kifejezések típusát nem változtatja meg. A harmadik lemma szerint egy kifejezés típusa nem változik, ha a típuskörnyezetből elhagyjuk a kötött változóira vonatkozó típusállításokat, azaz a típus nem változik, ha a típuskörnyezet csak a kifejezés szabad változóinak típusait tartalmazza.

Az E kifejezés struktúrájára alkalmazott indukcióval bizonyítható a következő három állítás:

3.6.1. Lemma. (Típuskörnyezet permutációja)

Ha $\Gamma \vdash E{:}A$ *és* Δ*-t a* Γ *elemeinek permutációjával állítjuk elő, akkor* $\Delta \vdash E{:}A$.

3.6.2. Lemma. (Típuskörnyezet kiterjesztése)

Ha $\Gamma \vdash E{:}A$ *és* $\Gamma \subseteq \Delta$*, akkor* $\Delta \vdash E{:}A$.

3.6.3. Lemma. (Típuskörnyezet szűkítése)

Ha $\Gamma \vdash E{:}A$ *és* $\Delta \equiv \{x{:}B \mid x{:}B \in \Gamma \text{ és } x \in FV(E)\}$*, akkor* $\Delta \vdash E{:}A$.

Most megmutatjuk azt, hogy ha adott egy kifejezés típusa, azaz a kifejezés jól típusozott, akkor ez a típus hogyan függ a kifejezés szerkezetétől.

3.6.4. Lemma. (Generálási lemma)

1. $$\frac{\Gamma \vdash x{:}A}{(x{:}A) \in \Gamma}$$
2. $$\frac{\Gamma \vdash EF{:}B}{\exists A \mid E{:}A \to B \text{ és } F{:}A}$$
3. $$\frac{\Gamma \vdash \lambda x{:}A\,.\,E{:}C}{\exists B \mid \Gamma, x{:}A \vdash E{:}B \text{ és } C \equiv A \to B}$$

A lemmában levő állítások a kifejezések típusára vonatkozó szabályok következményei.

Ahhoz, hogy megadhassuk az F_1 rendszer konverziós szabályait, először azt kell megadnunk, hogy hogyan határozhatjuk meg a típusos λ-kifejezések szabad változóit, majd a helyettesítés műveletét kell definiálnunk.

3.6.5. Definíció. A szabad változó:

1. *Az x változó szabad az x kifejezésben,*
2. *az x szabad az EF -ben, ha x szabad E -ben vagy szabad az F -ben,*
3. *az x szabad $\lambda y{:}A.E$ -ben, ha $x \neq y$ és az x szabad E -ben.*

Ha most is $FV(E)$ jelöli az E kifejezés szabad változóinak halmazát, akkor az 3.6.5. definíció alapján

- $FV(x) = \{x\}$,
- $FV(EF) = FV(E) \cup FV(F)$,
- $FV(\lambda x{:}A.E) = FV(E) \setminus \{x\}$,

Látható, hogy a szabad változók halmazának képzése megegyezik a típusnélküli kifejezéseknél megadott ?? definíció módszerével. A kötött változók, a zárt kifejezések, egy kifejezés lezárása is megegyezik a típusnélküli λ-kalkulusra megadott azonos nevű fogalmakkal.

Az $E[x := F]$ λ-kifejezéssel most is azt jelöljük, hogy az E-ben a szabad x változókat F-fel helyettesítjük. A *helyettesítést* az ?? Definícióhoz hasonlóan, most is az E szerkezete szerinti indukcióval adjuk meg.

3.6.6. Definíció. Helyettesítés:

1. $$x[y := G] \equiv \begin{cases} G, & \textit{ha } x \equiv y, \\ x, & \textit{egyébként}, \end{cases}$$
2. $$(EF)[y := G] \equiv (E[y := G])(F[y := G]),$$
3. $$(\lambda x{:}A.E)[y := G] \equiv \begin{cases} \lambda x{:}A.E, & \textit{ha } x \equiv y, \\ \lambda x{:}A.E[y := G], & \textit{ha } x \not\equiv y \textit{ és } x \notin FV(G), \\ \lambda x{:}A.E, & \textit{egyébként.} \end{cases}$$

Ha a helyettesítést tetszőleges y-ra és G-re hajtjuk végre, könnyen típushibát kaphatunk. Egy helyettesítés a típus szempontjából csak akkor lesz helyes, ha a helyettesítendő változó típusa megegyezik a behelyettesített kifejezés típusával. Ezt mondja ki a következő lemma:

3.6.7. Lemma. (Helyettesítési lemma)

$$\frac{\Gamma, x{:}A \vdash E{:}B \quad \Gamma \vdash F{:}A}{\Gamma \vdash E[x := F]{:}B} \quad [\textsc{Subst}]$$

A lemma azt is megadja, hogy az E kifejezés típusa a helyettesítéssel nem változik meg.

Ismerve egy típusos λ-kifejezés szabad változóit és a helyettesítés szabályait, megadhatjuk az α-konverzió definícióját is:

3.6.8. Definíció. Az α-konverzió:

Ha az E-ben nincs y szabad változó, akkor

$\lambda x{:}A.E \;\leftrightarrow_\alpha\; \lambda y{:}A.E[x := y]$.

Az α-konverzió a kifejezés típusát nem változtatja meg. Ezt mondja ki a következő, az α-konverzióra vonatkozó típusszabály.

3.6.9. Lemma. (Az α-konverzió típusszabálya)

$$\frac{\Gamma \vdash \lambda x{:}A.E{:}A \to B \quad y \notin FV(E) \quad y \notin dom(\Gamma)}{\Gamma \vdash \lambda y{:}A.E[x := y]{:}A \to B} \quad [\textsc{Conv}\ \alpha]$$

Most adjuk meg a típusos λ-kifejezésekre vonatkozó β-redukciót:

3.6.10. Definíció. A β-redukció:

Ha az $E[x := F]$-ben az F szabad változói nem válnak az E kötött változóivá, akkor

$(\lambda x{:}A.E)F \;\rightarrow_\beta\; E[x := F]$.

Talán meglepő, hogy a β-redukció végrehajtásakor nem vizsgáljuk az F kifejezés típusát, a redukciót bármilyen típusú F kifejezéssel végrehajthatjuk, azaz nem követeljük meg a redukálandó kifejezés jól típusozottságát.

Ha az F kifejezés típusa a fenti β-redukcióban nem A, akkor már a $(\lambda x{:}A.E)F$ kifejezés típushibás. A tárgyredukció tétele majd éppen azt mondja majd ki, hogy a jól típusozott kifejezések redukálásai nem okozhatnak típushibát.

Határozzuk meg a β-redukcióban szereplő két kifejezés típusát. Az $x : A$ és $E : B$ típusokból következik, hogy $\lambda x : A\,.\,E : A \to B$, és ha $F : A$, akkor $(\lambda x : A\,.\,E)F : B$. A helyettesítési lemma alapján ugyanezekből a feltételekből az $E[x := F]{:}B$ típust kapjuk. Ebből adódik a következő állítás:

3.6.11. Lemma. (A β-redukció típusszabálya)

Ha az $E[x := F]$-ben az F szabad változói nem válnak az E kötött változóivá, akkor

$$\frac{\Gamma \vdash (\lambda x{:}A.E)F{:}B}{\Gamma \vdash E[x := F]{:}B} \quad [\textsc{Conv}\ \beta]$$

A β-redukcióra vonatkozó típusszabályból kapjuk a következő tételt, ami azt mondja ki, hogy a β-redukciók sorozata egy jól típusozott kifejezés típusát nem változtatja meg.

3.6.12. Tétel. (A tárgyredukció tétele)

Ha $\Gamma \vdash E{:}A$ és $E \twoheadrightarrow^* F$, akkor $\Gamma \vdash F{:}A$.

A tétel elnevezése onnan származik, hogy egy $E{:}A$ típusállításban az E-t a típusállítás tárgyának nevezzük. A tétel csak a jól típusozott E kifejezésekre teljesül, azaz a $\Gamma \vdash E{:}A$ következtetésnek igaznak kell lennie.

Az állítás egyes esetekben fordítva is igaz.

3.6.13. Tétel. (A tárgykiterjesztés tétele)

Ha $\Gamma \vdash E{:}A$, $E \twoheadrightarrow^* F$ és $\Gamma \vdash F{:}B$, akkor $A \equiv B$.

A tárgykiterjesztés tételében az első feltétel, $\Gamma \vdash E{:}A$ lényeges. A feltétel azt jelenti, hogy az állítás csak akkor igaz, ha az E kifejezések van típusa.

Így tehát a tárgyredukció tételében a $\twoheadrightarrow^*$ iránya nem fordítható meg, azaz a tárgyredukció tétele szigorúan csak redukciókra vonatkozik. Mint a tárgykiterjesztés tétele mutatja, a tárgyredukció tétele β-absztrakciókra csak abban az esetben bővíthető, ha az E kifejezés típusa meghatározható.

Ha E nem típusozható, azaz típushibás, akkor $E \twoheadrightarrow^* F$ esetén az F kifejezésnek még lehet típusa. Ezt mutatja meg a következő példa.

3.6.14. Példa. *(A tárgyredukció tétele és a β-absztrakció)*

Nyilvánvaló, hogy $\mathsf{KI}\Omega \twoheadrightarrow_\beta \mathsf{I}$. A baloldalon álló kifejezés azonban típushibás, hiszen Ω-hoz semmilyen típus sem rendelhető, mint ezt a 3.3.6 példában láttuk.

$\square$

Ha egy kifejezésnek létezik típusa, akkor ez a típus egyértelmű.

3.6.15. Tétel. (Típusok egyértelműsége)

Ha $\Gamma \vdash E{:}A$ és $\Gamma \vdash E{:}B$, akkor $A \equiv B$.

Az F_1 típusrendszerben is megadhatjuk az η-konverziót, és meghatározhatjuk az η-konverzió típusszabályát is:

3.6.16. Definíció. Az η-konverzió:

Ha E egy λ-absztrakció, és az x az E-nek nem szabad változója, akkor $\lambda x : A . Ex \Leftrightarrow_\eta E$.

3.6.17. Lemma. (Az η-konverzió típusszabálya)

Ha x az E-nek nem szabad változója, akkor

$$\frac{\Gamma \vdash \lambda x{:}A.Ex{:}A \to B}{\Gamma \vdash E{:}A \to B} \quad [\textsc{Conv}\ \eta]$$

Látható, hogy a kifejezés típusát az η-konverzió sem változtatja meg.

A redukciós sorozat és a normál forma fogalma a típusnélküli azonos nevű fogalmakkal megegyező módon definiálhatók.

A típusnélküli λ-kalkulusban láttuk, hogy egy λ-kifejezésnek lehet, hogy nincs is normál formája. Most ezt a kérdést vizsgáljuk meg a típusos λ-kalkulusban.

Az F_1 rendszernek van egy olyan tulajdonsága, ami a típusnélküli rendszereknek nincs meg, és amelyik kulcsszerepet játszik az alkalmazásokban. Ez a következő állítás.

3.6.18. Tétel. (Az erős normalizálás tétele)

Az F_1 típusrendszerben az $E{:}A$ jól típusozott kifejezésnek nincs végtelen β-redukciós sorozata.

Ebből pedig azonnal következik a következő állítás:

3.6.19. Tétel. (A normalizálás tétele)

Az F_1 rendszerben minden jól típusozott λ-kifejezésnek van normál formája.

A típusos λ-kalkulusban is érvényes a Church-Rosser tétellel megfogalmazott rombusz-tulajdonság.

3.6.20. Tétel. (Az I. Church-Rosser tétel)

Ha az $E{:}A$ jól típusozott kifejezésre $E{:}A \twoheadrightarrow^* E_1{:}A$ és $E{:}A \twoheadrightarrow^* E_2{:}A$, akkor létezik egy olyan $F{:}A$ kifejezés, melyre $E_1{:}A \twoheadrightarrow^* F{:}A$ és $E_2{:}A \twoheadrightarrow^* F{:}A$.

A normalizálás tétele és az I. Chuch-Rosser tétel garantálja, hogy minden jól típusozott λ-kifejezésnek véges lépésben meghatározható a normál formája, és ez a normál forma egyértelmű. A tétel azt is kimondja, hogy a normál formára hozás lépéseiben a típus nem változik meg.

3.7. Az altípus

A modern programnyelvek egyik általános tulajdonsága az, hogy a típusrendszer altípusok kezelését is lehetővé teszi. Mint korábban láttuk, a típus egy értékhalmaz, ezért az A típus egy eleme a típus értékhalmazát tartalmazó bővebb halmaznak is eleme. Ez a bővebb halmaz is meghatároz egy típust, és ez a halmaz-tartalmazás vezet el az *altípus* fogalmához: az A típus a bővebb típus értékhalmaz által meghatározott B típus altípusa, ezt $A \leq B$-vel jelöljük.

Ha az F_1 rendszert kiegészítjük az altípus fogalmával, akkor ezt a kiegészített rendszert $F^1_{\leq}$ (ejtsd: ef-egy-szub) *rendszernek* nevezzük.

Az $F^1_{\leq}$ rendszer az F_1 rendszer bővítése, ezért csak az új szintaktikai elemet, az új következtetést és az új szabályokat adjuk meg.

3.7.1. Definíció. Az $F^1_{\leq}$ rendszer szintaxisa:

Az F_1 rendszernek a 3.2.1. definícióban megadott szintaxisa a következővel bővül:

$< típus >$ ::= *Top*

Az $F^1_{\leq}$ rendszerben tehát egy új alaptípus, a *Top* típus jelenik meg.

3.7.2. Definíció. $F^1_{\leq}$ rendszer következtetései:

Az F_1 rendszernek a 3.3.1. definícióban megadott következtetései a következővel bővülnek:

$\Gamma \vdash A \leq B$ az A típus a B típus altípusa

A következtetések érvényességére alkalmazható szabályok a következők:

3.7.3. Definíció. Az $F^1_{\leq}$ rendszer szabályai:

Az F_1 rendszernek a 3.3.2. definícióban megadott szabályai a következőkkel bővülnek:

A típusra vonatkozó új szabályok:

$$\frac{\Gamma \vdash wf}{\Gamma \vdash Top} \quad [\textsc{Type Top}]$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \leq Top} \quad [\textsc{Sub Top}]$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \leq A} \quad [\textsc{Sub Refl}]$$

$$\frac{\Gamma \vdash A \leq B \quad \Gamma \vdash B \leq C}{\Gamma \vdash A \leq C} \quad [\textsc{Sub Trans}]$$

$$\frac{\Gamma \vdash A' \leq A \quad \Gamma \vdash B \leq B'}{\Gamma \vdash A \to B \leq A' \to B'} \quad [\textsc{Sub Arrow}]$$

Kifejezés típusára vonatkozó új szabály:

$$\frac{\Gamma \vdash x : A \quad \Gamma \vdash A \leq B}{\Gamma \vdash x : B} \quad [\textsc{Val Subsumption}]$$

Az $F^1_{\leq}$ rendszerben tehát a *Top* típus a „legbővebb” típus, azaz minden jól típusozott kifejezés típusa a *Top*. Látható, hogy az altípus reláció reflexív és tranzitív, és a *Top* típus a maximális elem.

Alaptípusként a *Nat* és *Bool* típus mellett vezessük még be az *Int* típust, azaz az egész számok típusát és a *Real* típust, a valós számok típusát.

3.7.4. Példa. *(A Nat, Int és a Real típus)*
Nyilvánvalóan

$Nat \leq Top$, $Int \leq Top$, $Real \leq Top$,

és

$Nat \leq Int$, $Int \leq Real$,

és a Sub Trans szabály szerint a

$Nat \leq Real$

is fennáll.

Az is látható, hogy a „szűkebb” típus elemeinek több feltételt, követelményt kell kielégíteniük, mint a „bővebb” típus elemeinek, ezért a kisebb típus elemszáma is kisebb (vagy legalábbis nem nagyobb), mint a nagyobb típus elemeinek száma. □

A VAL SUBSUMPTION, azaz az *alárendelés* szabálya azt mondja ki, hogy ha egy kifejezés típusa A, és B az A-nál "bővebb" típus, azaz $A \leq B$, akkor a kifejezés típusának a B-t is megadhatjuk. Ez azt jelenti, hogy az altípus reláció a halmazokra megadható tartalmazás relációnak felel meg.

Az alárendelés szabálya a ***biztonságos helyettesítés elvét*** is leírja, ez az elv azt mondja ki, hogy $A \leq B$ esetén a B típus minden esetben biztonságosan helyettesíthető A-val, vagyis az A biztonságosan alkalmazható a B típus helyett.

3.7.5. Példa. *(Az alárendelés szabálya)*
Az alárendelés szabály egy alkalmazásának eredményeként azt a következtetést kapjuk, hogy ha az E kifejezés típusa Nat, akkor az $E{:}Int$ is fennáll. □

Két függvénytípus közötti altípusra vonatkozó SUB ARROW szabály a következőképpen magyarázható meg: Ha az E függvény típusa $A \to B$, akkor a függvény argumentumának típusa A, és argumentum lehet nyilván minden olyan kifejezés is, amelyiknek a típusa az A egy A' altípusa. Ugyanakkor, mivel a függvény értékének típusa B, a függvény értékének típusa egyben egy, a B-t tartalmazó bővebb B' típus is. Így az E függvény elfogadja az A' típusú argumentumot, és ad egy B' típusú eredményt, azaz típusa $A' \to B'$. Ez a tulajdonság azt jelenti, hogy minden $A \to B$ típus az $A' \to B'$ típus altípusa is, azaz $A \to B \leq A' \to B'$. Látható, hogy az altípus ***kontravariáns*** (***antimonoton***, ellenkező irányú) a függvények argumentumára, és ***kovariáns*** (***monoton***, azonos irányú) a függvények értékére.

3.7.6. Példa. *(Függvénytípusok közötti altípus reláció)*
Legyen az abszolútérték és a dekrementáló függvény a következő:

$\mathsf{abs}{:}Int \to Nat$,

$\mathsf{dec}{:}Nat \to Int$,

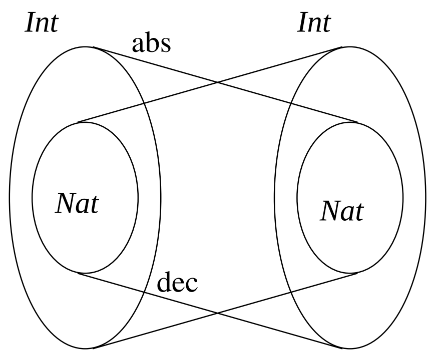
A két függvény típusára a SUB ARROW szabály szerint a

$Int \to Nat \leq Nat \to Int$

összefüggés írható fel (3.2. ábra), azaz az abszolútérték függvény típusa kisebb, mint a dekrementáló függvény típusa. □

3.7.7. Példa. *(Összetett függvények típusa)*
Legyenek ebben a példában az alaptípusok még a Rat, a racionális számok típusa, és $PosRat$, a pozitív racionális számok típusa. Legyen most $\mathsf{absp1}$ a racionális számokon értelmezett "abszolút érték + 1" függvény, így

$\mathsf{absp1}{:}Rat \to PosRat$.



3.2. ábra. Az altípus függvénytípusokra, $Int \to Nat \leq Nat \to Int$

Legyen továbbá div2 egy olyan függvény, ami egy egész szám felét adja eredményül, azaz

$\mathsf{div2}: Int \to Rat.$

Így az

$f \equiv \mathsf{absp1} \circ \mathsf{div2} \equiv \lambda x{:}Int\,.\,\mathsf{absp1}(\mathsf{div2}\ x)$

összetett függvény egy egész szám felének az abszolút értékét növeli 1-gyel, és $f: Int \to PosRat$.

Legyen az int függvény a valós számokon értelmezett egészrész függvény,

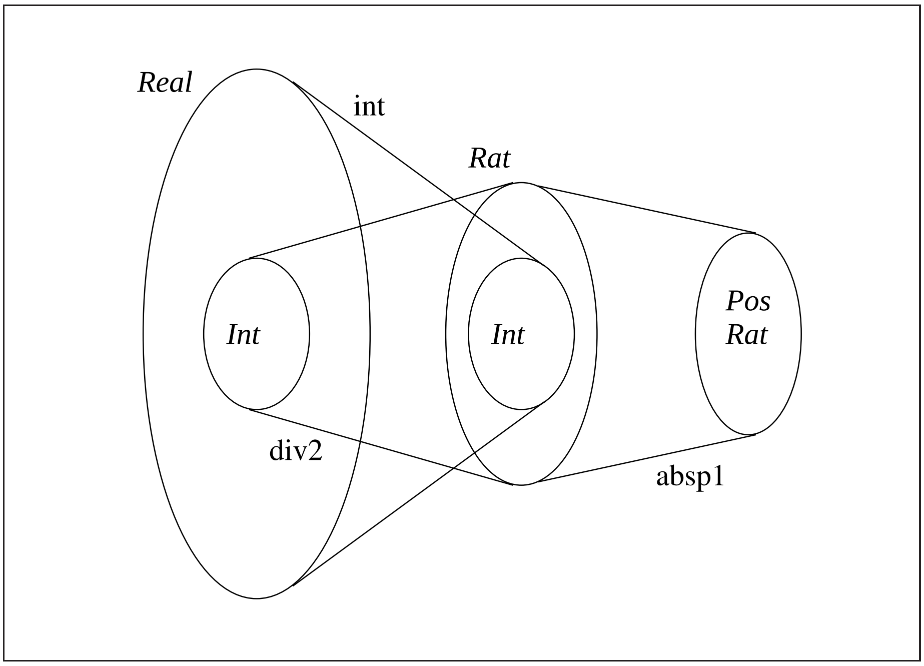
$\mathsf{int}: Real \to Int.$

Mivel $Real \to Int \leq Int \to Rat$, az int egészrész függvény típusa altípusa a div2 függvény típusának. Ez a Val Subsumption szabály szerint azt jelenti, hogy minden olyan kifejezésben, amelyben a div2 szerepel, a div2 függvényt biztonságosan le lehet cserélni az int-re, azaz a kettővel osztó függvény típushiba nélkül kicserélhető az egészrész függvényre. Így tehát az

$f' \equiv \mathsf{absp1} \circ \mathsf{int} \equiv \lambda x{:}Real\,.\,\mathsf{abs}(\mathsf{int}\ x)$

függvény is jól típusozott, és típusa $Real \to PosRat$ (3.3. ábra). □

Az altípus reláció megadható struktúrált típusokra is úgy, hogy az alárendelés Val Subsumption szabálya ezekre a típusokra is teljesül.



3.3. ábra. Függvénytípusok altípusainak alkalmazása

A *Pair* és a *Union* típusra az altípus az elemenkénti altípus figyelembevételével definiálható.

3.7.8. Definíció. A *Pair* típus altípus szabálya:

$$\frac{\Gamma \vdash A_1 \leq B_1 \quad \Gamma \vdash A_2 \leq B_2}{\Gamma \vdash Pair_{A_1 \times A_2} \leq Pair_{B_1 \times B_2}} \quad [\textsc{Sub Pair}]$$

3.7.9. Definíció. A *Union* típus altípus szabálya:

$$\frac{\Gamma \vdash A_1 \leq B_1 \quad \Gamma \vdash A_2 \leq B_2}{\Gamma \vdash Union_{A_1 + A_2} \leq Union_{B_1 + B_2}} \quad [\textsc{Sub Union}]$$

A *Record* típusra az altípus reláció az elemek darabszámának és az azonos címkéjű elemek típusának figyelembevételével határozható meg.

3.7.10. Definíció. A *Record* típus altípus szabálya:

$$\frac{\Gamma \vdash A_1 \leq B_1 \ \ldots \ \Gamma \vdash A_m \leq B_m \quad \Gamma \vdash A_{m+1} \ \ldots \ \Gamma \vdash A_n \quad (m \leq n)}{\{\!|\ l_1 : A_1, \ldots, l_n : A_n |\!\} \leq \{\!|\ l_1 : B_1, \ldots, l_m : B_m |\!\}}$$

[SUB RECORD]

A *Record* típusokra tehát a nagyobb elemszámú rekord lesz az altípusa a kisebb elemszámú rekordnak, aminek az az alapja, hogy a nagyobb elemszámú rekordban több "megkötés" van. Az azonos címkéjű elemekre természetesen az $A_i \leq B_i$ $(1 \leq i \leq m)$ elemenkénti altípus relációnak teljesülnie kell.

3.8. Az F_1 rendszerrel definiálható függvények

Az elsőrendű típusos λ-kalkulusban a rekurzív függvényekhez nem rendelhető típus, így az F_1 rendszer kifejező ereje a típusnélküli λ-kalkulusénál lényegesen kisebb, hiszen ott a parciális rekurzív függvények is leírhatók voltak. (lásd **??** pont).

Az F_1 rendszerrel a *kiterjesztett polinomok* írhatók le, ezek osztálya lényegesen kisebb, mint a rekurzív függvények osztálya.

Legyenek a *Nat* típus elemei a Church-számok, ebben a számjegyrendszerben az n természetes számot a következő típusos λ-kifejezés reprezentálja:

$$\mathsf{c_n} \equiv \lambda f\!:\!Nat \to Nat\,.\,\lambda x\!:\!Nat\,.\,f^n(x) \quad (n = 0, 1, 2, \dots).$$

A függvények típusának rövid leírására vezessük be a következő jelölést:

$$Nat^k \equiv Nat \to Nat \to \cdots \to Nat,$$

ahol a jobboldalon pontosan k darab *Nat* szerepel. Vizsgáljuk az $\mathbb{N}^n \to \mathbb{N}$ típusú függvények definiálhatóságát.

3.8.1. Definíció. λ-definiálható függvény:

Azt mondjuk, hogy az $f\!:\!\mathbb{N}^n \to \mathbb{N}$ függvény λ-definiálható, ha ezt a függvényt az

$$F\!:\!Nat^n \to Nat$$

λ-kifejezés reprezentálja, azaz minden $m_1, m_2, \dots m_n$ -re

$$F\ \mathsf{c}_{\mathsf{m_1}}\ \mathsf{c}_{\mathsf{m_2}}\ \dots\ \mathsf{c}_{\mathsf{m_n}}\ = \mathsf{c}_{f(m_1, m_2, \dots, m_n)}.$$

3.8.2. Példa. *(A konstans függvény)*
Az F_1 rendszerben a $K_m(x) = m$ konstans függvény λ-definiálható:

$$\mathsf{K_m} \equiv \lambda x\!:\!Nat\,.\,\mathsf{c}_m$$ □

3.8.3. Példa. *(Az előjel függvény)*
Legyen a $sg(x)$ előjel függvény a következő:

$$\begin{aligned} sg(0) &= 0, \\ sg(x) &= 1, \quad ha\ x \neq 0. \end{aligned}$$

Ekkor a $sg(x)$ függvény alakja az F_1 rendszerben:

$$\mathsf{sg} \equiv \lambda x\!:\!Nat\,.\,x(\mathsf{true}\ \mathsf{c_1})\ \mathsf{c_0} \equiv$$
$$\lambda x\!:\!Nat\,.\,x\underline{((\lambda x\!:\!Nat\,.\,\lambda y\!:\!Nat\,.\,x)\ \mathsf{c_1})}\ \mathsf{c_0} \ \twoheadrightarrow$$
$$\lambda x\!:\!Nat\,.\,x(\lambda y\!:\!Nat\,.\,\mathsf{c_1})\ \mathsf{c_0}.$$

Például, $\mathsf{sg}\ \mathsf{c_0} = \mathsf{c_0}$,

$\mathsf{sg}\ \mathsf{c}_0 \equiv$
$\underline{(\lambda x{:}Nat\,.\,x(\lambda y{:}Nat\,.\,\mathsf{c}_1)\ \mathsf{c}_0)\ \mathsf{c}_0} \twoheadrightarrow$
$\mathsf{c}_0(\lambda y{:}Nat\,.\,\mathsf{c}_1)\ \mathsf{c}_0 \equiv$
$\underline{(\lambda f{:}Nat \to Nat\,.\,\lambda x{:}Nat\,.\,x)(\lambda y{:}Nat\,.\,c_1)}\ \mathsf{c}_0 \twoheadrightarrow$
$\underline{(\lambda x{:}Nat\,.\,x)\ \mathsf{c}_0} \twoheadrightarrow$
c_0,

és $\mathsf{sg}\ \mathsf{c}_1 = \mathsf{c}_1$,

$\mathsf{sg}\ \mathsf{c}_1 \equiv$
$\underline{(\lambda x{:}Nat\,.\,x(\lambda y{:}Nat\,.\,\mathsf{c}_1)\ \mathsf{c}_0)\ \mathsf{c}_1} \twoheadrightarrow$
$\mathsf{c}_1(\lambda y{:}Nat\,.\,\mathsf{c}_1)\ \mathsf{c}_0 \equiv$
$\underline{(\lambda f{:}Nat \to Nat\,.\,\lambda x{:}Nat\,.\,fx)(\lambda y{:}Nat\,.\,\mathsf{c}_1)}\ \mathsf{c}_0 \twoheadrightarrow$
$\underline{(\lambda x{:}Nat\,.\,(\lambda y{:}Nat\,.\,\mathsf{c}_1)\ x)\ \mathsf{c}_0} \twoheadrightarrow$
$\underline{(\lambda y{:}Nat\,.\,c_1)\ \mathsf{c}_0} \twoheadrightarrow$
c_1. □

3.8.4. Példa. *(Az U_i^n szelektor függvény)*
Az F_1 rendszerben az $U_i^n(x_1, x_2, \dots, x_n) = x_i$ szelektor függvény λ-definiálható:

$\mathsf{U}_\mathsf{i}^\mathsf{n}(x_1, x_2, \dots, x_n) \equiv \lambda x_1{:}Nat\,.\,\lambda x_2{:}Nat\,.\,\dots\,.\,\lambda x_n{:}Nat\,.\,x_i$ □

3.8.5. Példa. *(Az összeadás és a szorzás művelet)*
Az F_1 rendszerben az összeadás és a szorzás művelete is megadható:

$\mathsf{add} \equiv \lambda x{:}Nat\,.\,\lambda y{:}Nat\,.\,\lambda p{:}Nat \to Nat\,.\,\lambda q{:}Nat\,.\,xp(ypq)$,
$\mathsf{mul} \equiv \lambda x{:}Nat\,.\,\lambda y{:}Nat\,.\,\lambda p{:}Nat \to Nat\,.\,x(yp)$

Ekkor például

$\mathsf{add}\ \mathsf{c}_0\ \mathsf{c}_1 \equiv$
$\underline{(\lambda x{:}Nat\,.\,\lambda y{:}Nat\,.\,\lambda p{:}Nat \to Nat\,.\,\lambda q{:}Nat\,.\,xp(ypq))\mathsf{c}_0}\ \mathsf{c}_1 \twoheadrightarrow$
$\underline{(\lambda y{:}Nat\,.\,\lambda p{:}Nat \to Nat\,.\,\lambda q{:}Nat\,.\,\mathsf{c}_0\ p(ypq))\mathsf{c}_1} \twoheadrightarrow$
$\lambda p{:}Nat \to Nat\,.\,\lambda q{:}Nat\,.\,\mathsf{c}_0\ p(\mathsf{c}_1 pq)) \equiv$
$\lambda p{:}Nat \to Nat\,.\,\lambda q{:}Nat\,.\,\underline{(\lambda f{:}Nat \to Nat\,.\,\lambda x{:}Nat\,.\,x)p}(\mathsf{c}_1 pq) \twoheadrightarrow$
$\lambda p{:}Nat \to Nat\,.\,\lambda q{:}Nat\,.\,\underline{(\lambda x{:}Nat\,.\,x)(\mathsf{c}_1 pq)} \twoheadrightarrow$
$\lambda p{:}Nat \to Nat\,.\,\lambda q{:}Nat\,.\,\mathsf{c}_1 pq \equiv$
$\lambda p{:}Nat \to Nat\,.\,\lambda q{:}Nat\,.\,\underline{(\lambda f{:}Nat \to Nat\,.\,\lambda x{:}Nat\,.\,fx)p}q \twoheadrightarrow$

$\lambda p{:}Nat \to Nat\,.\,\lambda q{:}Nat\,.\,\underline{(\lambda x{:}Nat\,.\,p)q} \;\twoheadrightarrow$
$\lambda p{:}Nat \to Nat\,.\,\lambda q{:}Nat\,.\,pq \equiv$
c_1,

és

$\mathsf{mul}\ \mathsf{c}_0\ \mathsf{c}_1 \equiv$
$\underline{(\lambda x{:}Nat\,.\,\lambda y{:}Nat\,.\,\lambda p{:}Nat \to Nat\,.\,x(yp))\mathsf{c}_0}\ \mathsf{c}_1 \;\twoheadrightarrow$
$\underline{(\lambda y{:}Nat\,.\,\lambda p{:}Nat \to Nat\,.\,\mathsf{c}_0(yp))\mathsf{c}_1} \;\twoheadrightarrow$
$\lambda p{:}Nat \to Nat\,.\,\mathsf{c}_0(\mathsf{c}_1 p) \equiv$
$\lambda p{:}Nat \to Nat\,.\,\underline{(\lambda f{:}Nat \to Nat\,.\,\lambda x{:}Nat\,.\,x)(\mathsf{c}_1 p)} \equiv$
$\lambda p{:}Nat \to Nat\,.\,\lambda x{:}Nat\,.\,x \equiv$
c_0. □

3.8.6. Definíció. A kiterjesztett polinomok:

Kiterjesztett polinomok a következő függvények:

1. $K_m(x) = m \quad (m = 1, 2, \dots)$ *konstans függvények,*

2. az $sg(x)$ *előjel függvény,*

3. az $U_i^n(x_1, x_2, \dots, x_n) = x_i \quad (1 \le i \le n)$ *szelektor függvény.*

Kiterjesztett polinomokból további új kiterjesztett polinomokat az összeadás és a szozás műveletével konstruálhatunk.

4. Az összeadás:
ha $f{:}\mathbb{N}^i \to \mathbb{N}$ *és* $g{:}\mathbb{N}^j \to \mathbb{N}$ *két kiterjesztett polinom,*
akkor az $f + g$ *kiterjesztett polinom legyen*
$(f + g)(x_1, \dots, x_i, y_1, \dots, y_j) = f(x_1, \dots, x_i) + g(y_1, \dots, y_j)$.

5. A szorzás:
ha $f{:}\mathbb{N}^i \to \mathbb{N}$ *és* $g{:}\mathbb{N}^j \to \mathbb{N}$ *két kiterjesztett polinom,*
akkor az $f \cdot g$ *kiterjesztett polinom legyen*
$(f \cdot g)(x_1, \dots, x_i, y_1, \dots, y_j) = f(x_1, \dots, x_i) \cdot g(y_1, \dots, y_j)$.

3.8.7. Tétel. (Schwichtenberg tétel [1976.])

Az F_1 *rendszerben definiálható függvények osztálya pontosan azonos a kiterjesztett polinomok osztályával.*

Az F_1 rendszer által leírható függvények halmaza kétféleképpen növelhető:

- a *Nat* típusnak nem a Church-számokat feleltetjük meg,
- az F_1 rendszerbe bevezetjük a *primitív rekurziót* és az *if-then-else* műveletet.

A $\mathsf{c_n}$ Church-szám

$\mathsf{c_n} \equiv \lambda f{:}Nat \to Nat\,.\,\lambda x{:}Nat\,.\,f^n(x) \quad (n = 0, 1, 2, \ldots),$

a szám típusa

$(Nat \to Nat) \to Nat \to Nat,$

nem ennyire speciális típusú számjegyekkel bővebb λ-definiálható függvényhalmazt kaphatunk.

A másik módszer az, hogy a típusok alaphalmazának a *Nat* és *Bool* típust választjuk, és az F_1 rendszert a *primitív rekurzió* és az *if-then-else* művelettel bővítjük. Ezt a bővített rendszert *Gödel T rendszerének* nevezzük, és λT-vel jelöljük.

A λT rendszerben *Nat* és *Bool* típusok, mint alaptípusok szerepelnek, és ezekre a típusokra alakalmazhatók az F_1 rendszer szabályai.

A rendszer konstansai a true és false konstansok, az if konstans, azaz δ-függvény, ezekre érvényesek a **??** pontban megadott szabályok:

$$\frac{\Gamma \vdash wf}{\Gamma \vdash \mathsf{true}{:}Bool} \quad [\textsc{Val True}]$$

$$\frac{\Gamma \vdash wf}{\Gamma \vdash \mathsf{false}{:}Bool} \quad [\textsc{Val False}]$$

$$\frac{\Gamma \vdash E{:}Bool \quad \Gamma \vdash F{:}A \quad \Gamma \vdash G{:}A}{\Gamma \vdash (\mathsf{if}\ E\ F\ G){:}A} \quad [\textsc{Fun If}]$$

A rendszernek két *Nat* típusú konstansa van, a $\mathsf{c_0}$ nulla konstans, és a succ δ-függvény, ezekre is érvényesek a **??** pontban leírt szabályok:

$$\frac{\Gamma \vdash wf}{\Gamma \vdash \mathsf{c_0}{:}Nat} \quad [\textsc{Val } c_0]$$

$$\frac{\Gamma \vdash E{:}Nat}{\Gamma \vdash \mathsf{succ}\ E{:}Nat} \quad [\textsc{Val Succ}]$$

A rendszerben van még egy δ-függvény, a recur konstans, ami a **??**. Definíció 5. pontjában f-fel jelölt primitív rekurzív függvény háromváltozós alakja:

$$\begin{aligned} f(0, x_2, x_3) &= g(x_2, x_3), \\ f(succ(x_1), x_2, x_3) &= h(f(x_1, x_2, x_3), x_1, x_2, x_3), \end{aligned}$$

és a λT rendszerben speciálisan

$g(x_2, x_3) = U_1^2(x_2, x_3) = x_2,$

és h a kétváltozós x_3 függvény, speciálisan

$h = x_3(u, v) = x_3(f(x_1, x_2, x_3), U_1^3(x_1, x_2, x_3)) = x_3(f(x_1, x_2, x_3), x_1)$,

tehát

$$\begin{array}{lcl} f(0, x_2, x_3) & = & x_2, \\ f(succ(x_1), x_2, x_3) & = & x_3(f(x_1, x_2, x_3), x_1). \end{array}$$

Most adjuk meg a λT rendszer operációs szemantikáját, a redukciókat $\twoheadrightarrow_T$ -vel jelöljük. A λT rendszerben a következő redukciók alkalmazhatók:

- β-redukció,
- az if-re vonatkozó redukció:
  $\mathsf{if\ true}\ E\ F \twoheadrightarrow_T E$
  $\mathsf{if\ false}\ E\ F \twoheadrightarrow_T F$
- a recur-ra vonatkozó redukció:
  $\mathsf{recur}\ \mathsf{c}_0\ x\ f \twoheadrightarrow_T x$
  $\mathsf{recur}\ (\mathsf{succ}\ n)\ x\ f \twoheadrightarrow_T f\ (\mathsf{recur}\ n\ x\ f)\ n$

A recur függvényre vonatkozó típusszabály a következő:

$$\frac{\Gamma \vdash E{:}Nat \quad \Gamma \vdash F{:}A \quad \Gamma \vdash G{:}A \to Nat \to A}{\Gamma \vdash (\mathsf{recur}\ E\ F\ G){:}A} \quad [\text{Fun Recur}]$$

A λT rendszerre is igaz az *erős normalizálás tétele*, és érvényes az *I. Church-Rosser tétel*, azaz a bevezetett két függvény az F_1 rendszer alapvető tulajdonságait nem változtatja meg.

Az if bevezetése után a and, or és not logikai függvények is könnyen definiálhatók:

$$\begin{array}{lcl} \mathsf{and}\ E\ F & \equiv & \mathsf{if}\ E\ F\ \mathsf{false} \\ \mathsf{or}\ E\ F & \equiv & \mathsf{if}\ E\ \mathsf{true}\ F \\ \mathsf{not}\ E & \equiv & \mathsf{if}\ E\ \mathsf{false\ true} \end{array}$$

Így például

$$\begin{array}{lcll} \mathsf{not\ true} & \equiv & \mathsf{if\ true\ false\ true} & \twoheadrightarrow_T \mathsf{false} \\ \mathsf{and\ false}\ F & \equiv & \mathsf{if\ false}\ F\ \mathsf{false} & \twoheadrightarrow_T \mathsf{false} \\ \mathsf{or\ true}\ F & \equiv & \mathsf{if\ true\ true}\ F & \twoheadrightarrow_T \mathsf{true} \end{array}$$

Az and és az or függvények argumentumainak sorrendje nem tetszőleges,

$$\begin{array}{ll} \mathsf{and}\ E\ \mathsf{false} & \not\twoheadrightarrow_T \mathsf{false} \\ \mathsf{or}\ E\ \mathsf{true} & \not\twoheadrightarrow_T \mathsf{true} \end{array}$$

mivel

$\mathsf{and}\ E\ \mathsf{false} \equiv \mathsf{if}\ E\ \mathsf{false}\ \mathsf{false}$
$\mathsf{or}\ E\ \mathsf{true} \equiv \mathsf{if}\ E\ \mathsf{true}\ \mathsf{true}$

és ezek nem redukálhatók tovább, mivel normál formában vannak.

A recur primitív rekurzió egy egyszerűbb alakja az *iteráció*, amit iter-rel jelölünk, és amelynek a redukciós szabályai a következők:

$\mathsf{iter}\ \mathsf{c}_0\ x\ f \twoheadrightarrow_T x$
$\mathsf{iter}\ (\mathsf{succ}\ n)\ x\ f \twoheadrightarrow_T f\ \mathsf{iter}(n\ x\ f)$

Az iter csak abban különbözik a recur függvénytől. hogy a harmadik argumentuma egy egyváltozós függvény. Ez az iter függvény megegyezik a 3.5. pontban is szereplő *iter* függvénnyel.

Látható, hogy az recur függvény egy magasabbrendű függvény, és ennek a magasabb rendű függvénynek a definiálása primitív rekurzióval történt. Lényegében ez a magasabb rendű recur függvény növeli meg a λT rendszer kifejező erejét.

Bizonyítható, hogy a *Bool* típusú konstansok és az if függvény nem növeli a λT rendszerben leírható függvények halmazát, így ezek csupán arra szolgálnak, hogy a λT rendszer szintaktikája változatosabb legyen.

Legyen $\mathcal{N} = < \mathbb{N}, +, *, 0, 1, = >$ az aritmetika standard modellje, ahol $+, *$ az operátorok, $0, 1$ a kitüntetett elemek, és $=$ egy reláció az $\mathbb{N}$ felett. $Th(\mathcal{N})$ olyan elsőrendű zárt formulák, azaz mondatok halmaza, amelyek igazak $\mathcal{N}$-ben. A *PA* Peano Aritmetika $Th(\mathcal{N})$ axiomatizálására szolgál, azaz arra, hogy $Th(\mathcal{N})$ minden mondata a *PA* következménye legyen.

A *PA* a 0 számból, a *succ* és az *isNumber* függvényekből, az öt Peano-axiómából és az operátorok működését megadó definíciókból áll.

A Peano-axiómák:

$(PA1)\ \ isNumber(0)$

$(PA2)\ \ \forall x\{isNumber(x) \Rightarrow isNumber(succ(x))\}$

$(PA3)\ \ \nexists x\{isNumber(x)\ \&\ 0 = succ(x)\}$

$(PA4)\ \ \nexists x \nexists y\{isNumber(x)\ \&\ isNumber(y)\ \&\ x \neq y\ \&\ succ(x) = succ(y)\}$

$(PA5)\ \ \Phi(0)\ \&\ \forall x\{isNumber(x)\ \&\ \Phi(x) \Rightarrow \Phi(succ(x))\} \Rightarrow \forall x\{isNumber(x) \Rightarrow \Phi(x)\}$

Az operátorok definíciója:

$(+1)\ \ +\ x\ 0 \twoheadrightarrow x$
$(+2)\ \ +\ x\ succ(y) \twoheadrightarrow succ(+\ x\ y)$
$(*1)\ \ *\ x\ 0 \twoheadrightarrow 0$
$(*2)\ \ *\ x\ succ(y) \twoheadrightarrow +\ (*\ x\ y)\ x$

Egy $f(\vec{x})$ függvény bizonyíthatóan teljes PA-ban, ha

$PA \ \vdash \ \forall\vec{x}\ \exists y\{t_f(\vec{x},y)=0\}$,

ahol t_f az f-t kiszámító algoritmus. Ez az állítás azt mondja ki, hogy a kiszámító algoritmus minden $\vec{x}$-re terminál.

3.8.8. Tétel. (A λT rendszerben definiálható függvények)

A λT rendszerben definiálható függvények osztálya megegyezik a Peano Aritmetikában bizonyíthatóan teljes függvények osztályával.

A λT rendszerben definiálható függvények osztálya még nem egyezik meg a rekurzív függvények osztályával, de például egy nem primitív rekurzív függvény, az *Ackermann-függvény* már leírható. Az *Ackermann-függvény* definíciója a következő:

$$\begin{array}{lcl} A(0,n) & = & n+1 \\ A(m+1,0) & = & A(m,1) \\ A(m+1,n+1) & = & A(m,A(m+1,n)) \end{array}$$

Az Ackermann-függvény néhány értéke az 1. táblázatban látható.

Írjuk át az Ackermann-függvény definícióját egyváltozós függvényekre:

$$\begin{array}{lcl} A_0(n) & = & n+1 \\ A_{m+1}(0) & = & A_m(1) \\ A_{m+1}(n+1) & = & A_m(A_{m+1}(n)) \end{array}$$

Az Ackermann-függvény λT-beli alakja legyen Ack, melyre $A(m,n) \equiv \mathsf{Ack}\ \mathsf{c_m}\ \mathsf{c_n}$, és így $A_m \equiv \mathsf{Ack}\ \mathsf{c_m}$, $A_{m+1} \equiv \mathsf{Ack}\ \mathsf{c_{m+1}}$.

Nyilvánvaló, hogy

$A_0 \equiv \mathsf{succ}$.

Az A_{m+1} pedig a következő alakban írható fel:

$A_{m+1} \equiv \lambda x\,.\,(\mathsf{recur}\ x(\mathsf{Ack}\ \mathsf{c_m c_1})\ (\lambda yz\,.\,\mathsf{Ack}\ \mathsf{c_m}\ y))$.

Ez megfelel az egyváltozós alakban megadott definíciónak, hiszen

$A_{m+1}(0) \equiv$

$\underline{(\lambda x\,.\,(\mathsf{recur}\ x(\mathsf{Ack}\ \mathsf{c_m c_1})(\lambda yz\,.\,\mathsf{Ack}\ \mathsf{c_m} y)))\mathsf{c_0}} \ \twoheadrightarrow_\beta$

$\mathsf{recur}\ \mathsf{c_0}\ (\mathsf{Ack}\ \mathsf{c_m c_1})(\lambda yz\,.\,\mathsf{Ack}\ \mathsf{c_m} y) \to_T$

$\mathsf{Ack}\ \mathsf{c_m c_1} \equiv$

$A_m(1)$,

és

$A_{m+1}(n+1) \equiv$

$\underline{(\lambda x\,.\,(\mathsf{recur}\ x(\mathsf{Ack}\ \mathsf{c_m}\mathsf{c_1})(\lambda yz\,.\,\mathsf{Ack}\ \mathsf{c_m}y)))(\mathsf{succ}\ \mathsf{c_n})} \ \to_\beta$

$\mathsf{recur}(\mathsf{succ}\ \mathsf{c_n})(\mathsf{Ack}\ \mathsf{c_m}\mathsf{c_1})(\lambda yz\,.\,\mathsf{Ack}\ \mathsf{c_m}y) \ \to_T$

$\underline{(\lambda yz\,.\,\mathsf{Ack}\ \mathsf{c_m}y)(\mathsf{recur}\ \mathsf{c_n}(\mathsf{Ack}\ \mathsf{c_m}\mathsf{c_1})(\lambda yz\,.\,\mathsf{Ack}\ \mathsf{c_m}y))}\mathsf{c_n} \ \to_\beta$

$\underline{(\lambda z\,.\,(\mathsf{Ack}\ \mathsf{c_m}(\mathsf{recur}\ \mathsf{c_n}(\mathsf{Ack}\ \mathsf{c_m}\mathsf{c_1})(\lambda yz\,.\,\mathsf{Ack}\ \mathsf{c_m}y))))\mathsf{c_n}} \ \to_\beta$

$\mathsf{Ack}\ \mathsf{c_m}(\mathsf{recur}\ \mathsf{c_n}(\mathsf{Ack}\ \mathsf{c_m}\mathsf{c_1})(\lambda yz\,.\,\mathsf{Ack}\ \mathsf{c_m}y)) \ \leftarrow_\beta$

$\mathsf{Ack}\ \mathsf{c_m}((\lambda x\,.\,(\mathsf{recur}\ x(\mathsf{Ack}\ \mathsf{c_m}\mathsf{c_1})(\lambda yz\,.\,\mathsf{Ack}\ \mathsf{c_m}y)))\mathsf{c_n}) \ \equiv$

$A_m(A_{m+1}(n))$.

Írjuk fel A_{m+1}-t egyszerűbb alakban. Legyen G olyan, hogy

$A_{m+1} \equiv G(\mathsf{Ack}\ \mathsf{c_m})$,

$G \equiv \lambda ux\,.\,((\mathsf{recur}\ x(u\mathsf{c_1})(\lambda yz.uy))$.

Ekkor Ack, az Ackermann-függvény λT-beli alakja

$\mathsf{Ack} \equiv \lambda x\,.\,(\mathsf{recur}\ x\ \mathsf{succ}(\lambda yz\,.\,Gy))$

3.8.9. Példa. *(Az $A(0,n)$ kiszámítása)*

$A(0,n) \equiv A_0(n) \equiv$

$\mathsf{Ack}\ \mathsf{c_0}\mathsf{c_n} \equiv \underline{(\lambda x\,.\,(\mathsf{recur}\ x\ \mathsf{succ}(\lambda yz\,.\,Gy)))\mathsf{c_0}}\mathsf{c_n} \ \to_\beta$

$(\mathsf{recur}\ \mathsf{c_0}\ \mathsf{succ}(\lambda yz\,.\,Gy))\mathsf{c_n} \ \to_T$

$\mathsf{succ}\ \mathsf{c_n} \equiv$

$n+1$ □

3.8.10. Példa. *(Az $A(m+1,0)$ kiszámítása)*

$A(m+1,0) \equiv$

$A_{m+1}(0) \equiv$

$\mathsf{Ack}\ (\mathsf{succ}\ \mathsf{c_m})\mathsf{c_0} \equiv$

$\underline{(\lambda x\,.\,(\mathsf{recur}\ x\ \mathsf{succ}(\lambda yz\,.\,Gy)))(\mathsf{succ}\ \mathsf{c_m})}\mathsf{c_0} \ \to_\beta$

$\mathsf{recur}(\mathsf{succ}\ \mathsf{c_m})\mathsf{succ}(\lambda yz\,.\,Gy)\mathsf{c_0} \ \to_T$

$\underline{(\lambda yz\,.\,Gy)\ (\mathsf{recur}\ \mathsf{c_m}\ \mathsf{succ}(\lambda yz\,.\,Gy))}\mathsf{c_m}\mathsf{c_0} \ \to_\beta$

$\underline{(\lambda z\,.\,G(\mathsf{recur}\ \mathsf{c_m}\ \mathsf{succ}(\lambda yz\,.\,Gy)))\ \mathsf{c_m}}\mathsf{c_0} \ \to_\beta$

$G(\mathsf{recur}\ \mathsf{c_m}\ \mathsf{succ}(\lambda yz\,.\,Gy))\mathsf{c_0} \leftarrow_\beta$

$G((\lambda x\,.\,(\mathsf{recur}\ x\ \mathsf{succ}(\lambda yz\,.\,Gy)))\mathsf{c_m})\mathsf{c_0} \equiv$

$G(\mathsf{Ack}\ \mathsf{c_m})\mathsf{c_0} \equiv$

$A_{m+1}(0)$,

erről pedig már beláttuk, hogy $A_m(1)$-gyel egyenlő. □

3.8.11. Példa. *(Az $A(m+1, n+1)$ kiszámítása)*
Az előző példában szereplő levezetéshez hasonlóan belátható, hogy

$A(m+1, n+1) \equiv$

$A_{m+1}(n+1) \equiv$

$\mathsf{Ack}\ (\mathsf{succ}\ (\mathsf{c_m})(\mathsf{succ}\ \mathsf{c_n}) \equiv$

$(\lambda x\,.\,(\mathsf{recur}\ x\ \mathsf{succ}(\lambda yz\,.\,Gy)))(\mathsf{succ}\ \mathsf{c_m})(\mathsf{succ}\ \mathsf{c_n}) =$

$A_{m+1}(n+1)$,

és már erről is beláttuk, hogy értéke $A_m(A_{m+1}(n))$. □

$A(m,n)$	$n=0$	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$
$m=0$	1	2	3	4	5	6
$m=1$	2	3	4	5	6	7
$m=2$	3	5	7	9	11	13
$m=3$	5	13	29	61	125	253
$m=4$	13	65333	$2^{65336}-3$	$2^{2^{65536}}-3$	$A(3,2^{2^{65536}}-3)$	$A(3,A(4,4))$
$m=5$	65533	$A(4,65533)$	$A(4,A(4,65536))$	$A(4,A(5,2))$	$A(4,A(5,3))$	$A(4,A(5,4))$
$m=6$	$A(4,65533)$	$A(5,A(4,65533))$	$A(5,A(6,1))$	$A(5,A(6,2))$	$A(5,A(6,3))$	$A(5,A(6,4))$

1. táblázat. Az Ackermann-függvény néhány értéke

4. FEJEZET

Curry-típusos λ-kalkulus

Az implicit típusos rendszer paradigmáját Curry adta meg 1934-ben [?] a kombinátor logikára, majd Curry és Feys [?], és később Curry, Hindley és Seldin [?] definiálta a típusnélküli λ-kalkulus kifejezéseire. Ebben a fejezetben először a Curry-típusos λ-kalkulust definiáljuk, majd megvizsgáljuk a Curry-típusos λ-kalkulus tulajdonságait, ezeket összehasonlítjuk a Church-típusos λ-kalkulus hasonló eredményeivel, végül a két típusos λ-kalkulus kapcsolatát elemezzük.

4.1. A Curry-típusrendszer

A Curry-típusos λ-kalkulust röviden Curry-típusrendszernek nevezzük. A Curry-típusrendszer típusai nem alaptípusokból, hanem ***típusváltozókból*** épülnek fel, ezért a Church-típusos λ-kalkulus, azaz az F_1 rendszer alaptípus halmazának elemeit – megkülönböztetésképpen – ***típuskonstansoknak*** is nevezhetjük. A típusváltozók konkrét típusértéket majd a *típuslevezetés* folyamán kapnak.

4.1.1. Definíció. A Curry-típusrendszer szintaxisa:

$$\begin{array}{lll}
< \textit{típus} > & ::= & < \textit{típusváltozó} > \\
 & | & (< \textit{típus} > \to < \textit{típus} >) \\
< \lambda\textit{-kifejezés} > & ::= & < \textit{változó} > \\
 & | & (\lambda \;\; < \textit{változó} > \;.\; < \lambda\textit{-kifejezés} >) \\
 & | & (< \lambda\textit{-kifejezés} > < \lambda\textit{-kifejezés} >)
\end{array}$$

A λ-kifejezésekre és a típusokra vonatkozó precedencia szabályok azonosak az F_1 rendszer megfelelő szabályaival.

Mint az 3.1. pontban láttuk, az F_1 rendszert röviden a

$\mathbb{T} = \mathbb{U} \mid \mathbb{T} \to \mathbb{T}$,

$\Lambda_{\mathbb{T}} = V \mid \lambda x{:}\mathbb{T} \,.\, \Lambda_{\mathbb{T}} \mid (\Lambda_{\mathbb{T}}\Lambda_{\mathbb{T}})$

alakban írtuk le, ahol $\mathbb{T}$ a típusokat, $\mathbb{U}$ az alaptípusokat, $\Lambda_{\mathbb{T}}$ a kifejezéseket jelölte. A Curry-típusos λ-kifejezések hasonló stílusú leírása a következő:

$\mathbb{T} = \mathbb{V} \mid \mathbb{T} \to \mathbb{T}$,

$\Lambda = V \mid \lambda x \,.\, \Lambda \mid (\Lambda\ \Lambda)$,

ahol $\mathbb{V}$ a típusváltozók, V a változók és Λ a kifejezések halmaza.

A típusváltozókat az $\alpha, \beta, \ldots$ betűkkel jelöljük, a típusváltozókat tartalmazó típuskifejezések jelölésére az $A, B, \ldots$ karaktereket használjuk. A típuskifejezéseket ebben a típusrendszerben ***típussémáknak*** is nevezhetjük.

Két típusváltozót csak akkor tekintünk azonosnak, ha nevük megegyezik. Így két típusséma akkor azonos, ha pontosan megegyeznek. Ezek szerint $\alpha \not\equiv \beta$, és $\alpha \to \alpha \not\equiv \beta \to \beta$.

A Γ típuskörnyezet típusváltozókat is tartalmazhat:

4.1.2. Definíció. A típuskörnyezet szintaktikája:

< *típuskörnyezet* >	::=	$\emptyset$
	|	< *típuskörnyezet* > , < *változó* > : < *típus* >
	|	< *típusváltozó* >

A típuskörnyezet < *változó* > : < *típus* > párosaira a 4.1.2. Definícióban szereplő megkötések itt is érvényesek.

A ***következtetések*** itt is $\Gamma \vdash \mathit{wf}$, $\Gamma \vdash A$ és $\Gamma \vdash E\!:\!A$ alakúak.

Mivel az $\Gamma \vdash E\!:\!A$ következtetésben az A tetszőleges $\mathbb{T}$-beli típust jelöl, a Curry-típusos λ-kalkulusban ***korlátozott polimorfizmusról*** beszélhetünk, ami a következőket jelenti.

Ha az A típussémában az $\alpha_1, \alpha_2, \ldots, \alpha_m$ típusváltozók szerepelnek, akkor ez a típusséma valójában a

$\forall\alpha_1\, \forall\alpha_2\, \ldots\, \forall\alpha_m \,.\, A$

kifejezésnek felel meg. A típussémákban a $\forall$ kvantorok ***kötik*** a típusváltozókat, és a típussémákban csak az univerzális kvantorok által ***kötött típusváltozók*** vannak. A típusváltozó ***hatásköre*** a teljes típusséma. A fenti típussémát röviden a vele ekvivalens $\forall\alpha_1\alpha_2 \ldots \alpha_m \,.\, A$ alakban is megadhatjuk.

Azt mondjuk, hogy egy $E\!:\!A$ következtetés ***érvényes*** a Γ típuskörnyezetben, azaz $\Gamma \vdash E\!:\!A$, ha bizonyítható a következő szabályok felhasználásával.

4.1.3. Definíció. A Curry-típusrendszer szabályai:

A környezetre vonatkozó szabályok:

$$\frac{}{\emptyset \vdash wf} \quad [\textsc{Env}\ \emptyset]$$

$$\frac{\Gamma \vdash A \quad x \notin dom(\Gamma)}{\Gamma, x\!:\!A \vdash wf} \quad [\textsc{Env}\ x]$$

A típusra vonatkozó szabályok:

$$\frac{\Gamma \vdash wf \quad \alpha \in \mathbb{V}}{\Gamma \vdash \alpha} \quad [\textsc{Type Var}]$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \to B} \quad [\textsc{Type Arrow}]$$

Kifejezés típusára vonatkozó szabályok:

$$\frac{\Gamma', x\!:\!A, \Gamma'' \vdash wf}{\Gamma', x\!:\!A, \Gamma'' \vdash x\!:\!A} \quad [\textsc{Val}\ x]$$

$$\frac{\Gamma, x\!:\!A \vdash E\!:\!B}{\Gamma \vdash \lambda x\,.\,E\!:\!A \to B} \quad [\textsc{Val Fun}]$$

$$\frac{\Gamma \vdash E\!:\!A \to B \quad \Gamma \vdash F\!:\!A}{\Gamma \vdash EF\!:\!B} \quad [\textsc{Val Appl}]$$

Az F_1 rendszer szabályaihoz hasonlóan, itt is látható, hogy a típusra vonatkozó szabályok (Type Var, Type Arrow) csak a szintaktikai szabályokat ismétlik meg, így, ha egy kifejezés A típusa jól formált, azaz szintaktikusan helyes, akkor a $\Gamma \vdash A$ következtetés mindig érvényes.

A Val Fun szabály feltételében levő következtetés típuskörnyezetében $\Gamma, x : A$ szerepel, ami azt jelenti, hogy az x nincs benne a Γ-ban, vagyis $x \notin dom(\Gamma)$. Látható, hogy a feltétel típuskörnyezetéből az $x\!:\!A$ itt is beépül a λ-absztrakcióba.

A Val Appl szabály $E\!:\!A \to B$ és $F\!:\!A$ feltételeiben az A mindkét helyen ugyanazt a típussémát jelöli, azaz az A-knak a jól típusozottsághoz azonosaknak kell lenniük. Ez azt jelenti, hogy ha például a $(\lambda x\,.\,x)y$ kifejezésben az x típusa $\alpha \to \alpha$ és $y\!:\!\beta$, akkor a kifejezés nem jól típusozott. A probléma a típusváltozó helyettesítésével oldható meg, ezzel majd a következő pontban foglalkozunk.

A Val Fun szabályt, mivel egy λ-kifejezésbe egy típust épít be, itt is „típusbevezetésnek”, a Val Appl szabályt pedig, mivel függvénytípusból egy egyszerű típust ad eredményül, „típuseliminációnak” is nevezzük.

4.1.4. Példa. *(Az $\mathsf{I} \equiv \lambda x\,.\,x$ kifejezés típusának meghatározása)*

$$\dfrac{\dfrac{\dfrac{\dfrac{\emptyset \vdash wf \quad \alpha \in \mathbb{V}}{\emptyset \vdash \alpha} \qquad x \notin dom(\emptyset)}{x{:}\alpha \vdash wf}}{x{:}\alpha \vdash x{:}\alpha}}{\emptyset \vdash \lambda x\,.\,x{:}\alpha \to \alpha}$$

□

4.1.5. Példa. *(Az $\mathsf{I} \equiv \lambda x\,.\,x$ kifejezés típusának meghatározása, egy másik lehetőség)*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\emptyset \vdash wf \quad \alpha \in \mathbb{V}}{\emptyset \vdash \alpha} \quad \dfrac{\emptyset \vdash wf \quad \alpha \in \mathbb{V}}{\emptyset \vdash \alpha}}{\emptyset \vdash \alpha \to \alpha} \qquad x \notin dom(\emptyset)}{x{:}\alpha \to \alpha \vdash wf}}{x{:}\alpha \to \alpha \vdash x{:}\alpha \to \alpha}}{\emptyset \vdash \lambda x\,.\,x{:}(\alpha \to \alpha) \to (\alpha \to \alpha)}$$

□

A példákból látható, hogy egy kifejezéshez több típust is le tudunk vezetni. Ezzel a problémával is a következő pontban foglalkozunk.

4.2. A Curry-típusrendszer operációs szemantikája

Először megmutatjuk azt, hogy ha adott egy kifejezés típusa, azaz a kifejezés jól típusozott, akkor ez a típus hogyan függ a kifejezés szerkezetétől.

4.2.1. Lemma. (Generálási lemma)

1. $\dfrac{\Gamma \vdash x{:}A}{(x{:}A) \in \Gamma}$

2. $\dfrac{\Gamma \vdash EF{:}B}{\exists A \mid E{:}A \to B \text{ és } F{:}A}$

3. $\dfrac{\Gamma \vdash \lambda x\,.\,E{:}C}{\exists A, B \mid \Gamma, x{:}A \vdash E{:}B \text{ és } C \equiv A \to B}$

A lemmában levő állítások a kifejezések típusára vonatkozó szabályok következményei.

Az F_1 rendszerben leírtakkal azonos módon definiálhatjuk a *szabad* és *kötött változókat* (3.6.5. Definíció) és a *helyettesítés* műveletet (3.6.6. Definíció).

Egy helyettesítés típushelyességét, az F_1 típusrendszerhez hasonlóan, a következő lemmával adhatjuk meg:

4.2.2. Lemma. (Helyettesítési lemma)

$$\frac{\Gamma, x{:}A \vdash E{:}B \quad \Gamma \vdash F{:}A}{\Gamma \vdash E[x := F]{:}B}$$

A lemma azt mondja ki, hogy egy helyettesítés a típus szempontjából csak akkor lesz helyes, ha a helyettesítendő változó típussémája megegyezik a behelyettesített kifejezés típussémájával. Az is látható, hogy a helyettesítés a kifejezés típussémáját nem változtatja meg.

A Curry-típusrendszerben a helyettesítést típusváltozókra is megadhatjuk.

4.2.3. Definíció. Típusváltozó helyettesítése:

1. $\alpha[\beta := B] \equiv \begin{cases} B, & \textit{ha } \alpha \equiv \beta, \\ \alpha, & \textit{egyébként,} \end{cases}$

2. $(A_1 \to A_2)[\beta := B] \equiv A_1[\beta := B] \to A_2[\beta := B].$

Láttuk, hogy a típussémákban a típusváltozókat implicit módon egy univerzális kvantor köti, de a változókra definiált „kötött" vagy „szabad változó" fogalmakhoz hasonlóakról itt nem beszélhetünk, hiszen a típusváltozókkal nem definiáltunk függvényeket. Ezért a típusváltozó helyettesítését a típusváltozó minden előfordulására el kell végeznünk. Tehát az $A[\alpha := B]$ azt jelöli, hogy az α típusváltozót az A típusban minden helyen a B típussal helyettesítjük. Természetesen, ha A-ban nincs α típusváltozó, akkor az $A[\alpha := B]$ helyettesítés az A-t nem változtatja meg.

A típushelyettesítés a kifejezés típusát megváltoztatja, az inkonzisztens, egyedi következetlen típushelyettesítések könnyen típushibához vezetnek. Ezért a típushelyettesítést definiálnunk kell a típuskörnyezetre is.

4.2.4. Definíció. Típusváltozó helyettesítése típuskörnyezetben:

Ha

$\Gamma \equiv \{(x_1{:}A_1), (x_2{:}A_2), \ldots, (x_n{:}A_n)\}$,

akkor

$\Gamma[\beta := B] \equiv \{(x_1{:}A_1[\beta := B]), (x_2{:}A_2[\beta := B]), \ldots, (x_n{:}A_n[\beta := B])\}.$

$\Gamma[\beta := B]$ tehát azt jelenti, hogy a Γ típuskörnyezet minden $x : A$ elemére végrehajtjuk az $A[\beta := B]$ helyettesítést.

A típuskörnyezetre végrehajtott helyettesítés a kifejezések típusát módosítja, de a jól típusozott kifejezések a helyettesítés után is jól típusozottak maradnak. Ezt mondja ki a következő lemma.

4.2.5. Lemma. (Típushelyettesítési lemma)

$$\frac{\Gamma \vdash E : A}{\Gamma[\alpha := B] \vdash E : A[\alpha := B]}$$

Az F_1 rendszerhez hasonlóan, a Curry-típusrendszerben is igaz a következő állítás:

4.2.6. Tétel. (A tárgyredukció tétele)

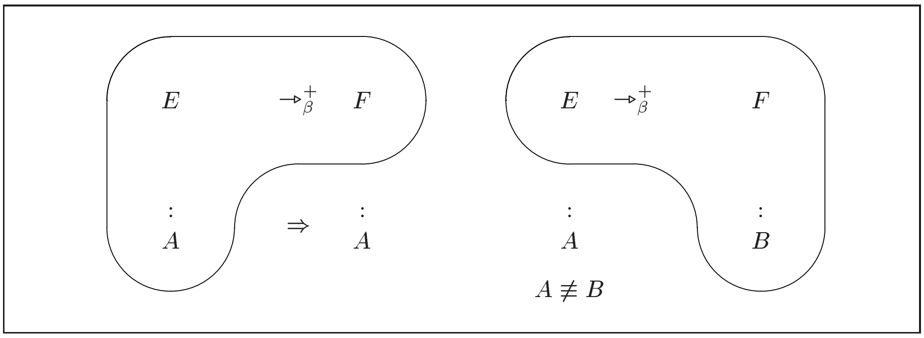
Ha $\Gamma \vdash E : A$ *és* $E \rightarrowtriangle^* F$, *akkor* $\Gamma \vdash F : A$.

Az F_1 rendszerben láttuk, hogy ha E és F mindegyike jól típusozott, akkor $E \rightarrowtriangle^* F$ esetén az E és F típusa β-absztrakciók alkalmazása esetén is megegyezik, azaz a tárgykiterjesztés tétele (4.2.7. Tétel) csak akkor áll fenn, ha a kifejezések jól típusozottak. A Curry-típusos λ-kalkulusban a tárgykiterjesztés tétele még a jól típusozott λ-kifejezésekre sem áll fenn, és ez a Church- és Curry-típusos kalkulusok közötti egyik lényeges különbség.

4.2.7. Tétel. (A tárgykiterjesztés tétele)

Ha $\Gamma \vdash E : A$, $E \rightarrowtriangle^* F$ *és* $\Gamma \vdash F : B$, *akkor* $A \not\equiv B$.

A tárgyredukció és a tárgykiterjesztés tétele szemléletesen a 4.1. ábrán látható.



4.1. ábra. A tárgyredukció és a tárgykiterjesztés tétele

A tárgykiterjesztés tételének bizonyítására a következő példát adjuk.

4.2.8. Példa. *(β-redukciók esetén a típusok nem feltétlenül azonosak)*
Könnyen belátható, hogy az

$\mathsf{S} \equiv \lambda xyz\,.\,xz(yz)$

kifejezés típusa

$(\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma,$

és a

$\mathsf{K} \equiv \lambda uv\,.\,u$

típusa

$\varphi \to \psi \to \varphi.$

A Val Appl szabály szerint az $\mathsf{S}\ \mathsf{K}$ kifejezés csak akkor típusozható, ha

$\alpha \to \beta \to \gamma \equiv \varphi \to \psi \to \varphi,$

azaz

$\alpha \equiv \gamma.$

Ha ezek az azonosságok fennállnak, akkor az

$\mathsf{S}\ \mathsf{K}\ \equiv (\lambda xyz\,.\,xy(xz))(\lambda uv\,.\,u)\ \twoheadrightarrow^{+}\ \lambda yz\,.\,z$

redukció mindegyik lépésében a típusok helyesek, és az eredményül kapott kifejezés típusa

$(\alpha \to \beta) \to \alpha \to \alpha.$

Ugyanakkor a $\lambda yz\,.\,z$ kifejezésre a

$\beta \to \alpha \to \alpha$

típus nyilvánvalóan levezethető, és ez a típus lényegesen különbözik az előzőtől.
$\square$

Az F_1 rendszerben, ha egy kifejezésnek létezett típusa, akkor ez a típus egyértelmű volt. A Curry-típusos λ-kalkulusban ez az állítás sem igaz, és ez okozza az előző példában megmutatott típusok különbözőségét.

4.2.9. Tétel. (A típusok nem egyértelműek)
Ha $\Gamma \vdash E{:}A$ és $\Gamma \vdash E{:}B$, akkor $A \not\equiv B$.

Ez a Church- és Curry-típusos kalkulusok közötti második lényeges különbség.

4.2.10. Példa. *(A $\lambda x\,.\,x$ kifejezés típusa nem egyértelmű)*

A 4.1.4. példában láttuk, hogy

$\lambda x\,.\,x{:}\alpha \to \alpha.$

Ugyanakkor a 4.1.5. példában azt kaptuk, hogy a

$\lambda x\,.\,x{:}(\beta \to \alpha) \to (\beta \to \alpha)$

is levezethető. □

4.3. A Church- és Curry-típusos λ-kalkulus kapcsolata

Bár a Church- és a Curry-típusos λ-kalkulus között nagyon sok különbség van, a két kalkulus között viszonylag egyszerűen lehet kapcsolatot teremteni. A kapcsolat lényege az, hogy a Church-típusos λ-kalkulus egy helyesen típusozott kifejezéséből az absztrakció változójának típusát elhagyva a Curry-típusos λ-kalkulus kifejezését kapjuk, és a Curry-típusos λ-kalkulus egy kifejezéséből a Church-típusos λ-kalkulus egy kifejezése lesz, ha az absztrakciók változóinak típusait megadjuk.

Jelöljük a Church-típusos λ-kalkulus kifejezéseinek halmazát $\Lambda_{\mathbb{T}}$-vel, a következtetéseket $\vdash_{Church}$-csel, a Curry-típusos λ-kalkulus kifejezéseit pedig Λ-val, a következtetéseket $\vdash_{Curry}$-val. Legyen $|\cdot|$ a következő, „felejtésnek”, vagy „törlésnek” is nevezhető függvény:

$|\cdot| : \Lambda_{\mathbb{T}} \Rightarrow \Lambda,$

ahol

$$\begin{array}{lll} |\,x\,| & \mapsto & x \\ |\,EF\,| & \mapsto & |\,E\,|\,|\,F\,| \\ |\,\lambda x : A.E\,| & \mapsto & \lambda x.\,|\,E\,| \end{array}$$

Látható, hogy ez a függvény a $\Lambda_{\mathbb{T}}$ kifejezéseit változtatja meg, az absztrakciók változóinak a típusát hagyja el, azaz a típusos λ-kifejezésekből típusnélküli kifejezések lesznek. A függvény definíciójából bizonyítható a következő két állítás.

4.3.1. Tétel. (Church → Curry konverzió)

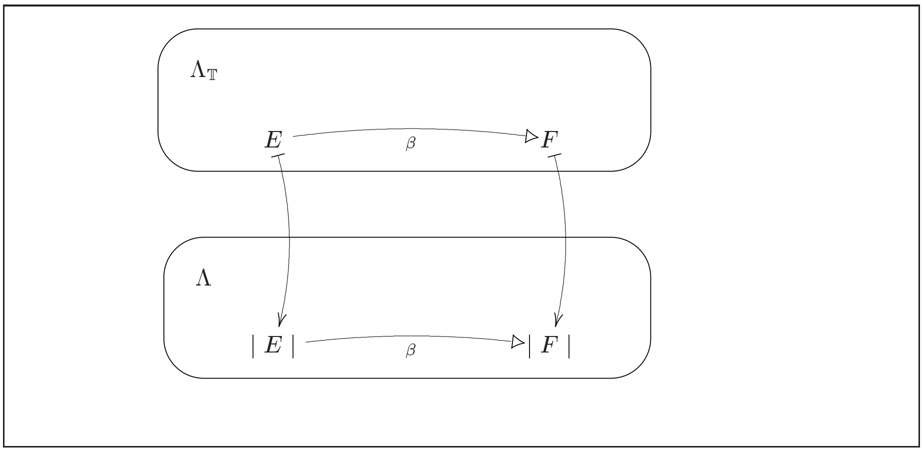
Legyen $E \in \Lambda_{\mathbb{T}}$. Ha $\Gamma \vdash_{Church} E{:}A$, akkor $\Gamma \vdash_{Curry} |\,E\,|{:}A$.

4.3.2. Tétel. (Curry → Church konverzió)

Legyen $E \in \Lambda$. Ha $\Gamma \vdash_{Curry} E{:}A$, akkor $\exists E' \in \Lambda_{\mathbb{T}}$, melyre $\Gamma \vdash_{Church} E'{:}A$, és $|\,E'\,| \mapsto E$.

Könnyen belátható az is, hogy a $|\cdot|$ függvény a kifejezések redukálását nem változtatja meg, azaz

- ha $E, F \in \Lambda_{\mathbb{T}}$ és $E \twoheadrightarrow_\beta F$, akkor $|E| \twoheadrightarrow_\beta |F|$ (4.2. bra),



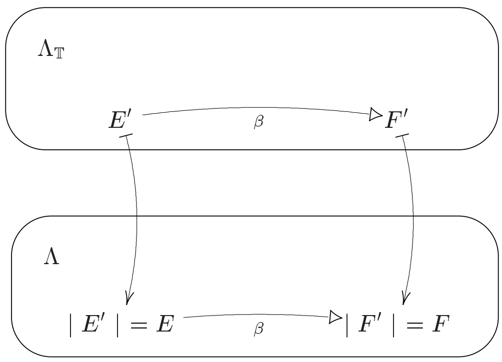
4.2. ábra. $\Lambda_{\mathbb{T}} \Rightarrow \Lambda$ és a $\Lambda_{\mathbb{T}}$ β-redukciója

$\lessdot Hd > [d]$

$Cons$

1 Nil

- ha $E, F \in \Lambda$ és $E \twoheadrightarrow_\beta F$, akkor minden $E' \in \Lambda_{\mathbb{T}}$-hez, melyre $|E'| \equiv E$, létezik olyan $F' \in \Lambda_{\mathbb{T}}$, amelyre $E' \twoheadrightarrow_\beta F'$ és $|F'| \equiv F$ (4.3. bra).



4.3. ábra. $\Lambda_{\mathbb{T}} \Rrightarrow \Lambda$ és a Λ β-redukciója

5. FEJEZET

Másodrendű (polimorfikus) típusos λ-kalkulus

Tekintsük a következő F_1-beli kifejezést:

$\mathsf{id\_Nat} \equiv \lambda x{:}Nat.x.$

Ez az *identitás* függvény a természetes számokra. Ha az identitás függvényt meg akarjuk adni *Int*, vagy *Bool* típusra, akkor olyan F_1-beli kifejezéseket kell megadnunk, amelyek lényegében megegyeznek a fenti kifejezéssel, csak a *Nat* helyett *Int*-et vagy *Bool*-t kell írnunk.

$$\begin{aligned} \mathsf{id\_Int} &\equiv \lambda x{:}Int.x, \\ \mathsf{id\_Bool} &\equiv \lambda x{:}Bool.x. \end{aligned}$$

Ez természetesen igaz tetszőleges típusra is, a típust mindig a *Nat* helyére kell írnunk.

Azért, hogy ne kelljen ilyen sok identitás függvényt megadnunk, jelöljük a típust egy α *típusváltozóval*:

$\mathsf{id\_}\alpha \equiv \lambda x{:}\alpha.x,$

ahol az α típusváltozó lehetséges értékei például a fenti típusok.

A típusváltozókat tartalmazó λ-kifejezéseket *típussémáknak* nevezzük.

A típusnélküli λ-kalkulusban egy λ-absztrakció x változója úgy kapott értéket, hogy először felírtunk egy applikációt, amelynek az első tagja egy λ-absztrakció, a második tagja, azaz az argumentum pedig az érték volt, majd erre az applikációra egy β-redukciót hajtottunk végre. Ehhez a módszerhez hasonlóan tudunk a típusváltozóhoz is értéket rendelni.

Először készítsünk az α típusváltozóval egy absztrakciót:

$\mathsf{id} \equiv \Lambda\alpha\,.\,\lambda x{:}\alpha\,.\,x,$

a típusváltozó absztrakcióját Λ-val jelöljük, azért nem λ-val, mert ehhez az absztrakcióhoz tartozó argumentum nem λ-kifejezés, hanem egy típus lesz.

Ezt az absztrakciót *típusabsztrakciónak*, a típusabsztrakcióval definiált függvényeket *polimorfikus függvényeknek* nevezzük. A fenti id kifejezés neve: polimorfikus identitás függvény.

Egy típusabsztrakcióval, mint függvénnyel és egy típussal, mint a függvény argumentumával képzett applikáció neve *típusapplikáció*.

Egy típusapplikációban a típusabsztrakció argumentumát [ ] zárójelbe tesszük, hangsúlyozva, hogy a kifejezésben ez egy típus, és nem λ-kifejezés.

A típusapplikáció β-redukcióját a λ-kifejezések applikációjának β-redukciójához hasonlóan definiálhatjuk, ekkor a típusabsztrakció az argumentumban megadott típust egy λ-kifejezésre képezi le.

Így az id típusabsztrakció az argumentumként adott típus identitás függvényét adja:

$$\mathsf{id}\ [Nat] \twoheadrightarrow_\beta \mathsf{id\_Nat},$$
$$\mathsf{id}\ [Int] \twoheadrightarrow_\beta \mathsf{id\_Int},$$
$$\mathsf{id}\ [Bool] \twoheadrightarrow_\beta \mathsf{id\_Bool},$$

és például

$$\mathsf{id}\ [Nat]\ 3 \equiv \underline{(\Lambda\alpha\,.\,\lambda x{:}\alpha\,.\,x)\ [Nat]}\ 3 \twoheadrightarrow_\beta \underline{(\lambda x{:}Nat\,.\,x)\ 3} \equiv \mathsf{id\_Nat}\ 3 \twoheadrightarrow_\beta 3.$$

Ha a típusos λ-kalkulust kiegészítjük a polimorfikus függvények kifejezéseivel, akkor meg kell adnunk e függvények típusát is. Mint láttuk, ezek a függvények a típust egy λ-kifejezésre képezik le. Az id függvény a Nat-ot egy $Nat \to Nat$ típusú függvényre, általánosan, az A típust egy $A \to A$ típusú függvényre képezi le, és látható, hogy az eredmény típusa csak az A-tól függ.

Ezt a tulajdonságot, azaz az id függvény típusát úgy jelöljük, hogy

$$\mathsf{id} : \forall\alpha\,.\,\alpha \to \alpha.$$

Megjegyezzük, hogy egy polimorfikus függvény $\forall\alpha\,.\,A$ típusát régebbi publikációkban $\lambda[\alpha]A$-val, $\Lambda\alpha\,.\,A$-val, $\Delta\alpha\,.\,A$-val, és $\forall[\alpha]A$-val is jelölték.

5.1. Az F_2 rendszer szintaxisa

A típusabsztrakcióval, típusapplikációval és a típusabsztrakció típusával bővített típusos λ-kalkulust *F_2 rendszernek*, *másodrendű típusos λ-kalkulusnak*, vagy *polimorfikus λ-kalkulusnak* nevezzük.

Az F_2 rendszer definícióját először, egymástól függetlenül Girard és Reynolds adta meg 1972-ben, illetve 1974-ben. Girard a logika oldaláról közelítette meg a témát, a Curry–Howard izomorfizmusnak az ítéletekre és a típusokra való kiterjesztésével, Reynold a programnyelvek paraméterátadási módszereiből indult ki, és a típus-paraméter bevezetése adta munkájához az alapgondolatot.

5.1.1. Definíció. Az F_2 rendszer szintaxisa:

$$
\begin{array}{lcl}
<\textit{típus}> & ::= & <\textit{típusváltozó}> \\
 & | & (<\textit{típus}> \to <\textit{típus}>) \\
 & | & (\forall <\textit{típusváltozó}> . <\textit{típus}>) \\
<\lambda\textit{-kifejezés}> & ::= & <\textit{változó}> \\
 & | & (\lambda\ <\textit{változó}>:<\textit{típus}> . <\lambda\textit{-kifejezés}>) \\
 & | & (<\lambda\textit{-kifejezés}> <\lambda\textit{-kifejezés}>) \\
 & | & (\Lambda <\textit{típusváltozó}> . <\lambda\textit{-kifejezés}>) \\
 & | & (<\lambda\textit{-kifejezés}>\ [<\textit{típus}>])
\end{array}
$$

A legkülső zárójelpárt elhagyhatjuk. A továbbiakban a típusváltozókat az $\alpha, \beta, \ldots$ görög kisbetűkkel jelöljük.

A fentiekben a típusabsztrakciót polimorfikus függvénynek neveztük, egy típusabsztrakció, azaz polimorfikus függvény típusát a továbbiakban röviden *polimorfikus függvénytípusnak* nevezzük. Tehát a $\forall\alpha\,.\,\alpha \to \alpha$ egy polimorfikus függvénytípus, az id polimorfikus függvénynek a típusa.

A $\Lambda\alpha\,.\,E$ típusabsztrakciónak, azaz polimorfikus függvénynek a típusát a $\forall\alpha\,.\,A$ típus írja le, ahol A az E λ-kifejezés típusa. Az α változót a polimorfikus függvénytípus *változójának* vagy *formális paraméterének*, az A típust a függvénytípus *törzsének* nevezzük. A $\forall\alpha\,.\,A$ polimorfikus függvénytípusban az A típus az α változó *hatásköre*, és azt mondjuk, hogy a hatáskörbe tartozó α változókat a polimorfikus függvénytípus α változója köti.

A polimorfikus függvénytípus $\forall\alpha\,.\,A$ leírásában az A típus nagyobb *precedenciájú*, mint a típusváltozó, így a redundáns zárójelpárok a törzsben elhagyhatók:

$$\forall\alpha\,.\,(\alpha \to \alpha) \equiv \forall\alpha\,.\,\alpha \to \alpha.$$

A $\Lambda\alpha\,.\,E$ típusabsztrakcióban az α változót a típusabsztrakció *változójának* vagy *formális paraméterének*, az E λ-kifejezést a típusabsztrakció *törzsének* nevezzük.

Az E kifejezés a típusabsztrakció α változójának a *hatásköre*, és azt mondjuk, hogy a hatáskörbe tartozó α változót a típusabsztrakció α változója köti. A típusabsztrakció *jobbasszociatív*, például, ha az E kifejezés is egy $\Lambda\beta\,.\,F$ típusabsztrakció, akkor $\Lambda\alpha\,.\,(\Lambda\beta\,.\,F)$ röviden így is írható:

$\Lambda\alpha\,.\,(\Lambda\beta\,.\,F) \equiv \Lambda\alpha\,.\,\Lambda\beta\,.\,F \equiv \Lambda\alpha\beta\,.\,F,$

azaz a redundáns zárójelpárok elhagyhatók, és a típusabsztrakció lambdái összevonhatók.

A típusapplikáció *balasszociatív*, így például

$(E[A])[B] \equiv E[A][B],$

azaz a redundáns zárójelpárok itt is elhagyhatók.

A típusapplikációnak nagyobb a *precedenciája*, mint a típusabsztrakciónak, ezt figyelembevéve a típusabsztrakció törzse így is írható:

$\Lambda\alpha\,.\,(E[A]) \equiv \Lambda\alpha\,.\,E[A].$

5.1.2. Példa. *(Polimorfikus függvénytípus)*

A 3.8. pontban láttuk, hogy 2-t reprezentáló Church-féle 2-es szám, vagy más szemlélettel a $\mathsf{double} \equiv \lambda fx\,.\,f(fx)$ függvény, F_1-beli alakja

$\lambda f{:}Nat \to Nat\,.\,\lambda x{:}Nat\,.\,f(fx)$

volt, és ehhez a függvényhez a

$(Nat \to Nat) \to Nat \to Nat$

típust rendeltük hozzá. Ha a Nat helyett az α típusváltozót használjuk, akkor

$\lambda f{:}\alpha \to \alpha\,.\,\lambda x{:}\alpha\,.\,f(fx),$

és ennek a típusa

$(\alpha \to \alpha) \to \alpha \to \alpha.$

A λ-kifejezésből típusabsztrakcióval a

$\Lambda\alpha\,.\,\lambda f{:}\alpha \to \alpha\,.\,\lambda x{:}\alpha\,.\,f(fx)$

polimorfikus függvényt kapjuk, melynek a típusa

$\forall\alpha\,.\,(\alpha \to \alpha) \to \alpha \to \alpha.$ □

5.2. Az F_2 típusrendszer

Az F_2 rendszer típusváltozókat is tartalmaz, ezért a Curry-típusos λ-kalkulushoz hasonlóan, a típuskörnyezetet ebben a rendszerben is a típusváltozokkal bővítjük.

5.2.1. Definíció. A típuskörnyezet szintaktikája:

$$\begin{array}{lcl} <\textit{típuskörnyezet}> & ::= & \emptyset \\ & | & <\textit{típuskörnyezet}>\ ,\ <\textit{változó}>:<\textit{típus}> \\ & | & <\textit{típusváltozó}> \end{array}$$

A típuskörnyezet $<$ *változó* $>:<$ *típus* $>$ párosaira a 4.1.2. Definícióban szereplő megkötések itt is érvényesek.

Az F_2 rendszer következtetései megegyeznek az F_1 rendszer következtetéseivel.

5.2.2. Definíció. Az F_2 típusrendszer következtetései:

$\Gamma \vdash \mathit{wf}$	Γ jól formált környezet
$\Gamma \vdash A$	Γ-ban az A jól formált típus
$\Gamma \vdash E{:}A$	Γ-ban az E kifejezés típusa A

A következtetések tehát formailag megegyeznek az F_1 rendszer következtetéseivel, de az F_2 rendszerben a Γ típuskörnyezet bővebb, mint az F_1-ben, hiszen itt a típuskörnyezet már típusváltozókat is tartalmaz.

5.2.3. Definíció. Az F_2 típusrendszer szabályai:

A környezetre vonatkozó szabályok:

$$\frac{}{\emptyset \vdash wf}\quad [\textsc{Env}\ \emptyset]$$

$$\frac{\Gamma \vdash A \quad x \notin dom(\Gamma)}{\Gamma, x{:}A \vdash wf}\quad [\textsc{Env}\ x]$$

$$\frac{\Gamma\ \vdash wf \quad \alpha \notin dom(\Gamma)}{\Gamma, \alpha \vdash wf}\quad [\textsc{Env}\ \alpha]$$

A típusra vonatkozó szabályok:

$$\frac{\Gamma', \alpha, \Gamma'' \vdash\ wf}{\Gamma', \alpha, \Gamma'' \vdash \alpha}\quad [\textsc{Type}\ \alpha]$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \to B}\quad [\textsc{Type Arrow}]$$

$$\frac{\Gamma, \alpha \vdash A}{\Gamma \vdash \forall\alpha\,.\,A}\quad [\textsc{Type Forall}]$$

Kifejezés típusára vonatkozó szabályok:

$$\frac{\Gamma', x{:}A, \Gamma'' \vdash wf}{\Gamma', x{:}A, \Gamma'' \vdash x{:}A}\quad [\textsc{Val}\ x]$$

$$\frac{\Gamma, x{:}A \vdash E{:}B}{\Gamma \vdash \lambda x{:}A\,.\,E{:}A \to B}\quad [\textsc{Val Fun}]$$

$$\frac{\Gamma \vdash E{:}A \to B \quad \Gamma \vdash F{:}A}{\Gamma \vdash EF{:}B}\quad [\textsc{Val Appl}]$$

$$\frac{\Gamma, \alpha \vdash E{:}A}{\Gamma \vdash \Lambda\alpha\,.\,E{:}\forall\alpha\,.\,A}\quad [\textsc{Val Fun2}]$$

$$\frac{\Gamma \vdash E{:}\forall\alpha\,.\,A \quad \Gamma \vdash B}{\Gamma \vdash E[B]{:}A[\alpha := B]}\quad [\textsc{Val Appl2}]$$

5.3. Típusok az F_2 típusrendszerben

Ebben az alfejezetben megvizsgáljuk, hogyan lehet az F_2 típusrendszerben ábrázolni a 3.5. alfejezetben bevezett néhány típust.

1. Az *Absurd* típus

A 3.5.2. példában adtuk meg az *Absurd* típus definícióját. Ez a típus az F_2

rendszerben nyilvánvalóan

$Absurd \equiv \forall\alpha\,.\,\alpha.$

Láttuk azt is, hogy a típusnak nem volt konstruktora. Ha most lenne egy ilyen konstruktor, az biztosan $\Lambda\alpha\,.\,E$ alakú, normál formájú zárt kifejezés lenne, de könnyen belátható, hogy nincs olyan E kifejezés, amellyel ez a konstruktor $Absurd$ típusú lenne.

2. A *Unit* típus

A $Unit$ típus (3.5.3. példa)

$Unit \equiv \forall\,.\,\alpha \to \alpha$

és a típuskonstruktor

$\mathsf{unit} \equiv \Lambda\alpha\,.\,\lambda x{:}\alpha\,.\,x.$

3. A *Bool* típus

Nyilvánvalóan, a $Bool$ típus

$Bool \equiv \forall\alpha\,.\,\alpha \to \alpha \to \alpha,$

és

$\mathsf{true} \equiv \Lambda\alpha\,.\,\lambda x{:}\alpha\,.\,\lambda y{:}\alpha\,.\,x,$

$\mathsf{false} \equiv \Lambda\alpha\,.\,\lambda x{:}\alpha\,.\,\lambda y{:}\alpha\,.\,y.$

Ekkor például a not függvénynek

$(\forall\alpha\,.\,\alpha \to \alpha \to \alpha) \to (\forall\beta\,.\,\beta \to \beta \to \beta)$

típusúnak kell lennie. Emlékeztetőül, a típusnélküli λ-kalkulusban

$\mathsf{not} \equiv \lambda x\,.\,x\ \mathsf{false}\ \mathsf{true}.$

Ez a not függvény az F_2 rendszerben a következőképpen adható meg:

$\mathsf{not} \equiv \lambda x{:}Bool\,.\,\Lambda\beta\,.\,\lambda y{:}\beta\,.\,\lambda z{:}\beta\,.\,x\,[\,\beta\,]\,zy.$

A not függvény valóban

$Bool \to (\forall\beta\,.\,\beta \to \beta \to \beta)$

típusú, mivel $x\,[\,\beta\,]\,zy$ típusa β.

5.3.1. Példa. *(A* not true *kifejezés)*
A not true kifejezés típusosan helyes, és

$\mathsf{not}\ \mathsf{true} \equiv$

$(\lambda x{:}Bool\,.\,\Lambda\beta\,.\,\lambda y{:}\beta\,.\,\lambda z{:}\beta\,.\,x\,[\,\beta\,]\,zy)(\Lambda\alpha\,.\,\lambda x{:}\alpha\,.\,\lambda y{:}\alpha\,.\,x) \ \twoheadrightarrow_\beta$
$\Lambda\beta\,.\,\lambda y{:}\beta\,.\,\lambda z{:}\beta\,.\,\underline{(\Lambda\alpha\,.\,\lambda x{:}\alpha\,.\,\lambda y{:}\alpha\,.\,x)\,[\,\beta\,]}\,zy \ \twoheadrightarrow_\beta$
$\Lambda\beta\,.\,\lambda y{:}\beta\,.\,\lambda z{:}\beta\,.\,(\lambda x{:}\beta\,.\,\lambda y{:}\beta\,.\,x)zy \ \twoheadrightarrow_\beta^+$
$\Lambda\beta\,.\,\lambda y{:}\beta\,.\,\lambda z{:}\beta\,.\,z \equiv$
false □

A not függvény egyszerűbben is megadható, ha a függvénybe beírjuk a true és false értékeket:

not $\equiv \lambda x{:}Bool\,.\,x\,[\,Bool\,]$ false true.

Hasonlóan, az and és or függvényekre

and $\equiv \lambda x{:}Bool\,.\,\lambda y{:}Bool\,.\,x\,[\,Bool\,]\,y$ false
or $\ \ \equiv \lambda x{:}Bool\,.\,\lambda y{:}Bool\,.\,x\,[\,Bool\,]$ true y

4. A *Nat* típus

A természetes számokat képviseljék most a Church-számok, amelyeknek alakja az F_2 típusrendszerben

$\mathsf{c_n} \equiv \Lambda\alpha\,.\,\lambda f{:}\alpha \to \alpha\,.\,\lambda x{:}\alpha\,.\,f^n(x)$.

Így a számok típusa, azaz a *Nat* típus

$Nat \equiv \forall\alpha\,.\,(\alpha \to \alpha) \to \alpha \to \alpha$.

A succ függvény típusának nyilvánvalóan $Nat \to Nat$ típusnak kell lennie, a succ $\equiv \lambda nfx\,.\,f(nfx)$ függvényre a következő kifejezés adható meg:

succ $\equiv \lambda n{:}Nat\,.\,\Lambda\alpha\,.\,\lambda f{:}\alpha \to \alpha\,.\,\lambda x{:}\alpha\,.\,f(n\,[\,\alpha\,]\,fx)$.

Mivel az $f(n\,[\,\alpha\,]\,fx)$ kifejezés típusa α, a succ függvény típusa valóban $Nat \to Nat$.

Hasonlóan belátható, hogy a Church-számokra megadott műveletek is leírhatók az F_2 típusrendszerben:

zero $\equiv \lambda x{:}Nat(\lambda y{:}Bool\,.$ false)true
add $\ \equiv \lambda x{:}Nat\,.\,\lambda y{:}Nat\,.\,\Lambda\alpha\,.\,\lambda p{:}\alpha \to \alpha\,.\,\lambda q{:}\alpha\,.\,x\,[\,\alpha\,]\,p(ypq)$,
mul $\ \equiv \lambda x{:}Nat\,.\,\lambda y{:}Nat\,.\,\lambda p{:}\alpha\,.\,x\,[\,\alpha\,]\,(yp)$,
exp $\ \equiv \lambda x{:}Nat\,.\,\lambda y{:}Nat\,.\,y\,[\,\alpha\,]\,x$.

5. A *Pair* típus

A pair kifejezés alakja a típusnélküli λ-kalkulusban $\lambda xyz.zxy$ volt. Az F_2 típusrendszerben ezt a következőképpen írhatjuk le:

$\mathsf{Pair}_{\alpha\times\beta} \equiv \forall\gamma\,.\,(\alpha \to \beta \to \gamma) \to \gamma,$

és a pair, first és second függvények

$$\begin{array}{ll}
\mathsf{pair}_{\alpha\to\beta} & \equiv \lambda x{:}\alpha\,.\,\lambda y{:}\beta\,.\,\forall\gamma\,.\,\lambda z{:}\alpha \to \beta \to \gamma\,.\,zxy \\
\mathsf{first} & \equiv \lambda x{:}Pair_{\alpha\to\beta}\,.\,x\,[\,\alpha\,]\,(\lambda y{:}\beta\,.\,\lambda z{:}\gamma\,.\,x) \\
\mathsf{second} & \equiv \lambda x{:}Pair_{\alpha\to\beta}\,.\,x\,[\,\beta\,]\,(\lambda y{:}\beta\,.\,\lambda z{:}\gamma\,.\,z)
\end{array}$$

5.4. Az F_2 rendszerrel definiálható függvények

A λT rendszerben a primitív rekurzió definiálható volt. Vizsgáljuk meg, hogy a primitív rekurzió leírható-e az F_2 rendszerben.

6. FEJEZET

Speciális típusos λ-kalkulusok

6.1. Az egzisztenciális típus

Az F_2 típusrendszer VAL APPL2 típusszabálya azt mondja ki, hogy ha az $E : \forall\alpha\,.\,A$ polimorfikus kifejezésre a B típust applikáljuk, akkor az $E\,[\,B\,]$ kifejezés típusa $A\,[\,\alpha := B\,]$.

Ez azt jelenti, hogy például az $\mathsf{id\_Nat} \equiv \lambda x : Nat\,.\,x$ kifejezés típusát az $\mathsf{id}\,[\,Nat\,] \equiv (\Lambda : \alpha\,.\,\lambda x : \alpha\,.\,x)\,[\,Nat\,]$ kifejezés típusaként is előállíthatjuk. Mivel az id típusa $\forall\alpha\,.\,\alpha \to \alpha$, az $\mathsf{id\_Nat}$ kifejezés típusa $(\alpha \to \alpha)\,[\,\alpha := Nat\,]$. Tehát az $\alpha \to \alpha$ típushoz van olyan típus, a Nat, amelyet az α típusváltozóba helyettesítve az $\mathsf{id\_Nat}$ típusát kapjuk meg.

Legyen tehát $F : \forall\alpha\,.\,A$, és legyen $F\,[\,B\,] \twoheadrightarrow_\beta E$. Ekkor $E : A\,[\,\alpha := B\,]$, azaz ha az α-t, az A típus típusváltozóját a B típussal helyettesítjük, az E kifejezés típusát kapjuk meg. Tehát az A-beli α típusváltozóra van olyan B típus, amellyel az E típusa leírható.

Ezt a tulajdonságot fejezzük ki a $\exists\,\alpha\,.\,A$ *egzisztenciális típussal.* A $\exists$ egy operátor, azaz típuskonstruktor, felhasználásával írjuk le az egzisztenciális típust. A $\exists\,\alpha\,.\,A$ típusban az α típusváltozó *hatásköre* a A típuskifejezés.

A Curry-típusos λ-kalkulus és az F_2 típusrendszer az egzisztenciális típus bevezetésével bővíthető.

6.1.1. Definíció. A típusra vonatkozó szintaktikus szabályok:

$$
\begin{array}{rcl}
<\textit{típus}> & ::= & <\textit{típusváltozó}> \\
& | & (\ <\textit{típus}> \to <\textit{típus}>\ ) \\
& | & (\ \forall <\textit{típusváltozó}>\ .\ <\textit{típus}>\ ) \\
& | & (\ \exists <\textit{típusváltozó}>\ .\ <\textit{típus}>\ )
\end{array}
$$

Mivel az egzisztenciális típust az $E : A\,[\,\alpha := B\,]$ kifejezésből vezettük le, a $\exists\,\alpha\,.\,A$ típus egy *eleme* a B és E kifejezésekkel reprezentálható. A $\exists\,\alpha\,.\,A$ típus egy elemét tehát leírhatjuk a $< B, E >$ párral, de mi következő kifejezőbb jelölést fogjuk használni:

$\mathsf{pack}\ B\ \mathsf{with}\ E.$

Ezt a kifejezést a továbbiakban *csomagnak* nevezzük. Az elnevezés arra utal, hogy az $\alpha := B$ típus-értékadást „becsomagoljuk”, elrejtjük a típus használói elől.

A $\exists\,\alpha\,.\,A$ egzisztenciális típus elemeire gyakran a

$\mathsf{pack}_{\exists\,\alpha\,.\,A}\ \alpha = B\ \mathsf{with}\ E$

jelölést, vagy a

$\mathsf{pack}\ \alpha = B\ \mathsf{as}\ E$

kifejezést is használják.

6.1.2. Példa. *(Az egzisztenciális típus)*
Az $\mathsf{id\_Nat} \equiv \lambda x{:}Nat\,.\,x$ kifejezés típusa $Nat \to Nat$, ez a típus a VAL FUN2 szabály alapján az $\mathsf{id}\,[\,Nat\,]$ kifejezés típusa. Mivel

$\mathsf{id} \equiv (\Lambda\alpha\,.\,\lambda x{:}\alpha\,.\,x){:}\forall\alpha\,.\,\alpha \to \alpha,$

az $\mathsf{id\_Nat}$ típusa

$(\alpha \to \alpha)\,[\,\alpha := Nat\,]$

alakban is felírható, azaz

$(\lambda x{:}Nat\,.\,x)\ :\ (\alpha \to \alpha)\,[\,\alpha := Nat\,].$

Ebből a

$(\mathsf{pack}\ Nat\ \mathsf{with}\ \lambda x{:}Nat\,.\,x)\ :\ \exists\,\alpha\,.\,\alpha \to \alpha$

kifejezést kapjuk meg. □

6.1.3. Példa. *(Az egzisztenciális típus)*
A $\lambda x{:}Nat\,.\,x + 1$ kifejezés típusa $Nat \to Nat$, ami nyilvánvalóan az

$(\alpha \to \alpha)\,[\,\alpha := Nat\,]$

alakban is felírható, azaz

$(\lambda x{:}Nat\,.\,x + 1)\ :\ (\alpha \to \alpha)\,[\,\alpha := Nat\,].$

Ebből a

$(\mathsf{pack}\ Nat\ \mathsf{with}\ \lambda x{:}Nat\,.\,x + 1)\ :\ \exists\,\alpha\,.\,\alpha \to \alpha$

kifejezést írhatjuk fel. Látható, hogy a csomag típusa megegyezik az előző példában szereplő, ettől a csomagtól különböző csomag típusával. □

6.1.4. Példa. *(Az egzisztenciális típus)*
A $\lambda x{:}Bool\,.\,1$ kifejezés típusa $Bool \to Nat$, amiből az előző példában leírtakhoz hasonlóan,

$(\lambda x{:}Bool\,.\,1) : (\alpha \to Nat)\,[\,\alpha := Bool\,]$.

Ebből

$(\textsf{pack}\ Bool\ \textsf{with}\ \lambda x{:}Bool\,.\,1) : \exists\,\alpha\,.\,\alpha \to Nat.$ □

6.1.5. Példa. *(Az egzisztenciális típus elemei)*

- Nem minden szintaktikusan helyes egzisztenciális típusnak van „értelme”, például a $\exists\,\alpha\,.\,\alpha$ típus egy eleme $\textsf{pack}\ Nat\ \textsf{with}\ 0$, ami új információt nem tartalmaz, hiszen a nyilvánvaló

  $0{:}\alpha\,[\,\alpha := Nat\,]$

  kifejezést írja le.
- Hasonlóan, a $\exists\,\alpha\,.\,Nat \to Nat$ típus sem hordoz információt, mivel a típuskifejezésben nem szerepel az α típusváltozó. A típus egy eleme például $\textsf{pack}\ Bool\ \textsf{with}\ \textsf{succ}$, ami a

  $\textsf{succ}{:}Nat \to Nat[\,\alpha := Bool\,]$

  kifejezésnek felelne meg.
- Egy egzisztenciális típusnak nyilvánvalóan több különböző eleme lehet, például a $\exists\,\alpha\,.\,\alpha \to Nat$ típusnak eleme a következő két kifejezés:

  $\textsf{pack}\ Nat\ \textsf{with}\ \lambda x{:}Nat\,.\,x$,

  $\textsf{pack}\ Bool\ \textsf{with}\ \lambda x{:}Bool\,.\,\textsf{if}\ x\ 0\ 1$.
- Több különböző egzisztenciális típusnak lehet közös eleme, például a $\exists\,\alpha\,.\,\alpha \to \alpha$, $\exists\,\alpha\,.\,Nat \to \alpha$ és a $\exists\,\alpha\,.\,\alpha \to Nat$ típusok mindegyikének eleme a

  $\textsf{pack}\ Nat\ \textsf{with}\ \lambda x{:}Nat\,.\,x$

  kifejezés. □

Az egzisztenciális típus elemeit, a csomagokat „megnyitásuk”, „kicsomagolásuk” után használhatjuk, ezt a műveletet az $\textsf{unpack}$ kulcsszóval jelöljük. Az

$\textsf{unpack}\ \beta, x\ \textsf{from}\ F\ \textsf{in}\ G$

műveletben az F egy $\exists\,\alpha\,.\,A$ típusú pakolt $\textsf{pack}\ B\ \textsf{with}\ E$ kifejezés. A művelet definiál egy új β lokális típusváltozót, amit az $\textsf{unpack}$ köt, és a típusváltozó hatásköre az G kifejezés. Ez azt is jelenti, hogy a $\beta \notin FV(G)$ állításnak a típushelyesség érdekében teljesülnie kell. A művelet ezenkívül definiál egy új x változót is, amelyre $x{:}A\,[\,\alpha := \beta\,]$.

A művelet végrehajtásakor az $x := E$ értékadás hajtódik végre, de az E típusában, mint az x típusa is mutatja, az α típusváltozók β-ra cserélődnek. Ezután a G kiértékelésekor az x komponenseire lehet hivatkozni, az x komponensei az E komponenseinek felelnek meg. Ez tehát azt jelenti, hogy egy $F \equiv (\mathsf{pack}\ B\ \mathsf{with}\ E){:}\exists\,\alpha\,.\,A$ alkalmazásakor az α típusváltozó, annak B értéke és az E kifejezés is rejtve marad.

A fenti unpack műveletet egyes publikációkban az

$\mathsf{open}\ F\ \mathsf{as}\ \beta, x\ \mathsf{in}\ G,$

vagy a

$\mathsf{let}\ (\beta, x) = F\ \mathsf{in}\ G$

kifejezéssel is jelölik.

Összefoglalva az eddigieket, megadjuk a típusrendszer szintaktikájának új bővítéseit és új szabályait.

6.1.6. Definíció. A kifejezésekre vonatkozó szintaktikus szabályok:

$$
\begin{array}{lcl}
<\lambda\text{-}\textit{kifejezés}> & ::= & <\textit{változó}> \\
 & \mid & (\ \lambda\ <\textit{változó}>:<\textit{típus}>\ .\ <\lambda\text{-}\textit{kifejezés}>\ ) \\
 & \mid & (\ <\lambda\text{-}\textit{kifejezés}>\ <\lambda\text{-}\textit{kifejezés}>\ ) \\
 & \mid & (\ \Lambda <\textit{típusváltozó}>\ .\ <\lambda\text{-}\textit{kifejezés}>\ ) \\
 & \mid & (\ <\lambda\text{-}\textit{kifejezés}>\ [\ <\textit{típus}>\ ]\ ) \\
 & \mid & \mathsf{pack}\ <\textit{típus}>\ \mathsf{with}\ <\lambda\text{-}\textit{kifejezés}> \\
 & \mid & \mathsf{unpack}\ <\textit{típusváltozó}>\ ,\ <\textit{változó}> \\
 & & \quad \mathsf{from}\ <\lambda\text{-}\textit{kifejezés}>\ \mathsf{in}\ <\lambda\text{-}\textit{kifejezés}>
\end{array}
$$

6.1.7. Definíció. Az egzisztenciális típusra vonatkozó szabályok:

$$\frac{\Gamma, \alpha \vdash A}{\Gamma \vdash \exists\,\alpha\,.\,A}\quad [\textsc{Type Exists}]$$

$$\frac{\Gamma \vdash B \quad \Gamma \vdash E{:}A\,[\,\alpha := B\,]}{\Gamma \vdash\ \mathsf{pack}\ B\ \mathsf{with}\ E{:}\exists\,\alpha\,.\,A}\quad [\textsc{Val Pack}]$$

$$\frac{\Gamma \vdash \beta \quad \Gamma \vdash F{:}\exists\,\alpha\,.\,A \quad \Gamma, x{:}A\,[\,\alpha := \beta\,] \vdash G{:}C \quad \beta \notin FV(C)}{\Gamma \vdash (\mathsf{unpack}\ \beta, x\ \mathsf{from}\ F\ \mathsf{in}\ G){:}C}\quad [\textsc{Val Unpack}]$$

A pack és unpack műveletek operációs szemantikáját redukciókkal adjuk meg, ezeket a redukciókat is a $\twoheadrightarrow_\beta$ jellel jelöljük.

6.1.8. Definíció. Az egzisztenciális típus operációs szemantikája:

Ha $E : A[\,\alpha := B\,]$, *és így* $\mathsf{pack}\ B\ \mathsf{with}\ E{:}\exists\,\alpha\,.\,A$, *akkor*

$\mathsf{unpack}\ \beta, x\ \mathsf{from}\ (\mathsf{pack}\ B\ \mathsf{with}\ E)\ \mathsf{in}\ G\ \twoheadrightarrow_\beta$

$G[\,x := E\,][\,\beta := B\,]$.

6.1.9. Példa. *(Az* unpack *művelet)*
A 6.1.3. példában láttuk, hogy

$(\mathsf{pack}\ Nat\ \mathsf{with}\ \lambda x{:}Nat\,.\,x+1) : \exists\,\alpha\,.\,\alpha \to \alpha$

Jelöljük a fenti kifejezést E-vel, ekkor az

$\mathsf{unpack}\ \beta, y\ \mathsf{from}\ E\ \mathsf{in}\ (y\ 3)$

kifejezés típusosan helyes, és

$\mathsf{unpack}\ \beta, y\ \mathsf{from}\ E\ \mathsf{in}\ (y\ 3)\ \twoheadrightarrow_\beta$
$(y\ 3)[\,y := \lambda x{:}Nat\,.\,x+1\,][\,\beta := Nat] \equiv$
$(y\ 3)[\,y := \lambda x{:}Nat\,.\,x+1\,] \equiv$
$(\lambda x{:}Nat\,.\,x+1)\ 3\ \twoheadrightarrow^+$
4

Látható, hogy a típusváltozó helyettesítésére nem is volt szükség. □

6.1.10. Példa. *(Az* unpack *művelet)*
Az előző példában szereplő

$E \equiv (\mathsf{pack}\ Nat\ \mathsf{with}\ \lambda x{:}Nat\,.\,x+1) : \exists\,\alpha\,.\,\alpha \to \alpha$

kifejezésre redukáljuk az

$\mathsf{unpack}\ \beta, y\ \mathsf{from}\ E\ \mathsf{in}\ ((\lambda z{:}\beta\,.\,y\ z)\ 3)$

kifejezést. A kifejezés típusosan helyes, így

$\mathsf{unpack}\ \beta, y\ \mathsf{from}\ E\ \mathsf{in}\ ((\lambda z{:}\beta\,.\,y\ z)\ 3)\ \twoheadrightarrow_\beta$
$((\lambda z{:}\beta\,.\,y\ z)\ 3)[\,y := \lambda x{:}Nat\,.\,x+1\,][\,\beta := Nat] \equiv$
$((\lambda z{:}Nat\,.\,y\ z)\ 3)[\,y := \lambda x{:}Nat\,.\,x+1\,] \equiv$
$(\lambda z{:}Nat\,.\,(\lambda x{:}Nat\,.\,x+1)\ z)\ 3\ \twoheadrightarrow_\beta$
$(\lambda x{:}Nat\,.\,x+1)\ 3\ \twoheadrightarrow^+$
4

Ugyanerre az E kifejezésre az

$\mathsf{unpack}\ \beta, y\ \mathsf{from}\ E\ \mathsf{in}\ (\lambda z{:}\beta\,.\,y\ z)\ \mathsf{true}$

kifejezés típushibás, mivel a $(\lambda z:\beta\,.\,y\ z)\ \mathsf{true}$ kifejezésből látható, hogy $\beta \equiv \mathit{Bool}$, de az unpack művelet operációs szemantikája szerint az E-ből a $\beta := \mathit{Nat}$ helyettesítést kapjuk. □

A következőkben megmutatjuk, hogy hogyan tudjuk felhasználni az egzisztenciális típust a programozásban. Példákat adunk arra, hogy az absztrakt adattípusok használatakor az egzisztenciális típus hogyan biztosítja a típus reprezentációjának elrejtését, és megmutatjuk, hogy a felhasználó a programjában hogyan tudja használni ezt a típust.

6.1.11. Példa. *(A Szám1 típus)*
Definiáljunk egy *Szám1* típust úgy, hogy

$\mathit{Szám1} \equiv \mathit{Nat}$,

és erre a típusra adjunk meg három függvényt. A függvények definícióit, szignatúráit foglaljuk egy *Record* típusba:

$$\begin{aligned} A_{\mathit{Szám1}} \equiv \{\!|\ & \mathsf{make} : \mathit{Nat} \to \mathit{Szám1}, \\ & \mathsf{add} : \mathit{Szám1} \to \mathit{Szám1} \to \mathit{Szám1}, \\ & \mathsf{parity} : \mathit{Szám1} \to \mathit{Bool} \\ |\!\}& \end{aligned}$$

A függvények implementációja, azaz ennek a típusnak egy eleme legyen a következő rekord:

$$\begin{aligned} E_{\mathit{Szám1}} \equiv \{\ & \mathsf{make} = \lambda x{:}\mathit{Nat}\,.\,x, \\ & \mathsf{add} = \lambda x{:}\mathit{Szám1}\,.\,\lambda y{:}\mathit{Szám1}\,.\,\mathsf{add}\ x\ y, \\ & \mathsf{parity} = \lambda x{:}\mathit{Szám1}\,.\,\mathsf{isodd}\ x \\ \}& \end{aligned}$$

ahol add a *Szám1* típuson értelmezett összeadás művelete, isodd pedig egy olyan függvény, amely true értéket ad, ha az argumentuma páratlan szám, és értéke false, ha az argumentum páros.

Ha az $A_{\mathit{Szám1}}$ típusban a *Szám1* típust az α típusváltozóval helyettesítjük és az így kapott típust A-val jelöljük, azaz

$$\begin{aligned} A \equiv \{\!|\ & \mathsf{make} : \mathit{Nat} \to \alpha, \\ & \mathsf{add} : \alpha \to \alpha \to \alpha, \\ & \mathsf{parity} : \alpha \to \mathit{Bool} \\ |\!\}& \end{aligned}$$

akkor

$A_{Szám1} \equiv A\,[\,\alpha := Szám1\,]\,,$

és így

$E_{Szám1}\!:\!A\,[\,\alpha := Szám1\,]\ .$

Ebből pedig azt kapjuk, hogy

pack $Szám1$ with $E_{Szám1}\!:\!\exists\,\alpha\,.\,A.$ □

6.1.12. Példa. *(A Szám2 típus)*
Definiáljunk egy *Szám2* típust a *Pair* típussal úgy, hogy a pár első eleme legyen egy *Nat* típusú szám, a második eleme pedig egy *Bool* érték, ami true akkor, ha a pár első eleme páratlan, és false, ha az első elem páros.

$Szám2 \equiv Pair_{Nat\times Bool}.$

Erre a típusra is adjunk meg az előző példában szereplő három függvényt, a függvények definícióit most is foglaljuk egy *Record* típusba:

$$\begin{array}{lll} A_{Szám2} & \equiv \{\!| & \mathsf{make} : Nat \to Szám2, \\ & & \mathsf{add} : Szám2 \to Szám2 \to Szám2, \\ & & \mathsf{parity} : Szám2 \to Bool \\ & |\!\} & \end{array}$$

A függvények implementációja, azaz ennek a típusnak egy eleme legyen a következő rekord:

$$\begin{array}{lll} E_{Szám2} & \equiv \{ & \mathsf{make} = \lambda x\!:\!Nat\,.\,<x, \mathsf{isodd}\ x>, \\ & & \mathsf{add} = \lambda x\!:\!Szám2\,.\,\lambda y\!:\!Szám2\,. \\ & & \quad <\mathsf{add}(\mathsf{first}\ x)(\mathsf{first}\ y), \mathsf{xor}(\mathsf{second}\ x)(\mathsf{second}\ y)>, \\ & & \mathsf{parity} = \lambda x\!:\!Szám2\,.\,\mathsf{second}\ x \\ & \}, & \end{array}$$

ahol add a *Szám2* típuson értelmezett összeadás művelete, isodd az előző példában is szereplő, a *Nat* típuson értelmezett függvény, xor pedig az *exclusive or* műveletet valósítja meg.

Ha az $A_{Szám2}$ típusban a *Szám2* típust az α típusváltozóval helyettesítjük, az előző példában szereplő A kifejezést kapjuk, azaz most is

$$\begin{array}{lll} A & \equiv \{\!| & \mathsf{make} : Nat \to \alpha, \\ & & \mathsf{add} : \alpha \to \alpha \to \alpha, \\ & & \mathsf{parity} : \alpha \to Bool \\ & |\!\} & \end{array}$$

Tehát

$A_{Szám2} \equiv A\,[\,\alpha := Szám2\,]\,,$

és így

$E_{\mathit{Szám2}}{:}A\,[\,\alpha := \mathit{Szám2}\,]$.

Ebből pedig a $\exists\,\alpha\,.\,A$ típus egy új, az előző példában levő elemétől különböző elemét kapjuk,

pack $\mathit{Szám2}$ with $E_{\mathit{Szám2}}{:}\exists\,\alpha\,.\,A$. □

6.1.13. Példa. *(A felhasználói program)*
Tegyük fel, hogy két *Nat* típusú szám összegének a paritását akarjuk meghatározni, a feladat legyen például a következő *Bool* típusú kifejezés értékének a meghatározása:

parity(add(make 28)(make 32)).

Ismerjük a programban szereplő függvények definícióját, tehát a programfüggvény:

$$
\begin{array}{lll}
\mathsf{client} \equiv & \Lambda\alpha\,. & \\
& \lambda x{:}\{\!|\ \mathsf{make} & : \ \mathit{Nat}\to\alpha, \\
& \qquad \mathsf{add} & : \ \alpha\to\alpha\to\alpha, \\
& \qquad \mathsf{parity} & : \ \alpha\to\mathit{Bool} \\
& \quad |\!\}\ . & \\
& \mathsf{let}\ \{\!|\ \mathit{makes} & : \ \mathit{Nat}\to\alpha, \\
& \qquad \mathit{adds} & : \ \alpha\to\alpha\to\alpha, \\
& \qquad \mathit{calpar} & : \ \alpha\to\mathit{Bool} \\
& \quad |\!\} & \\
& \quad \{\ \mathit{makes} & = \ x\,.\,\mathsf{make}, \\
& \qquad \mathit{adds} & = \ x\,.\,\mathsf{add}, \\
& \qquad \mathit{calpar} & = \ x\,.\,\mathsf{parity} \\
& \quad \} & \\
& \mathsf{in}\ \mathit{calpar}(\mathit{adds}(\mathit{makes}\ 28)(\mathit{makes}\ 32)), &
\end{array}
$$

ahol *makes*, *adds* és *calpar* lokális változók, az importált modul függvényei kerülnek ide.

Használhatjuk a *Szám1* számábrázolást az $E_{\mathit{Szám1}}$ kifejezéssel,

client $[\,\mathit{Szám1}\,]\ E_{\mathit{Szám1}}$,

vagy a *Szám2* számábrázolást az $E_{\mathit{Szám2}}$ kifejezéssel,

client $[\,\mathit{Szám2}\,]\ E_{\mathit{Szám2}}$.

Még a client függvény argumentumait sem kell megadni, ha bevezetünk egy *Client* típust, aminek a típusértéke a fenti client kifejezés, és megadunk két

„könyvtári modult betöltő” kifejezést a következőképpen:

$\mathsf{lib}_1 \equiv \lambda x\!:\!\mathit{Client}\,.\,x\,[\,\mathit{Szám1}\,]\;\;E_{\mathit{Szám1}}$

$\mathsf{lib}_2 \equiv \lambda x\!:\!\mathit{Client}\,.\,x\,[\,\mathit{Szám2}\,]\;\;E_{\mathit{Szám2}}$

Ekkor a programunk a

lib_1 client

vagy a

lib_2 client

kifejezésekre egyszerűsödik. □

6.1.14. Példa. *(A felhasználói program az egzisztenciális típussal)*

A 6.1.13. példa egy nagyon egyszerű megoldást ad a típusok implementációjának elrejtésére, a probléma azonban az, hogy a lib_1 és lib_2 kifejezések argumentuma egy polimorfikus függvény, és ez a típuslevezetésben könnyen nemdeterminisztikusságot okozhat. Emiatt a polimorfikus argumentum használatát, azaz az előbbi példában szereplő megoldási módszert több programnyelvben nem is teszik lehetővé.

Az általános megoldás a fenti feladatra az egzisztenciális típus használata. Mint az előző 6.1.11. és 6.1.12. példákban láttuk, készítsük el a *Szám1* típusból és az $E_{\mathit{Szám1}}$ kifejezésből, valamint a *Szám2* típusból és az $E_{\mathit{Szám2}}$ kifejezésből a következő csomagokat:

$p_1 \equiv \mathsf{pack}\ \mathit{Szám1}\ \mathsf{with}\ E_{\mathit{Szám1}},$

$p_2 \equiv \mathsf{pack}\ \mathit{Szám2}\ \mathsf{with}\ E_{\mathit{Szám2}}.$

Az is láttuk, hogy mindkét csomag típusa

$$\exists\,\alpha\,.\ \{\!|\ \begin{array}{lcl} \mathsf{make} & : & \mathit{Nat}\to\alpha, \\ \mathsf{add} & : & \alpha\to\alpha\to\alpha, \\ \mathsf{parity} & : & \alpha\to\mathit{Bool} \end{array} \\ |\!\}$$

Legyen ez a két csomag két imlementációs programmodul, és ekkor ezeknek a moduloknak az exportálását az *unpack* kifejezéssel írhatjuk le. Ha a p_1 implementációt használjuk, akkor a program a következő lesz:

$$
\begin{array}{l}
\mathsf{unpack}\ \beta, x\ \mathsf{from}\ p_1\ \mathsf{in} \\
\qquad \mathsf{let}\ \{\!|\ \begin{array}{lcl} \mathit{makes} & : & Nat \to \beta, \\ \mathit{adds} & : & \beta \to \beta \to \beta, \\ \mathit{calpar} & : & \beta \to Bool \end{array} \\
\qquad\quad |\!\} \\
\qquad\quad \{\ \begin{array}{lcl} \mathit{makes} & = & x\,.\,\mathsf{make}, \\ \mathit{adds} & = & x\,.\,\mathsf{add}, \\ \mathit{calpar} & = & x\,.\,\mathsf{parity} \end{array} \\
\qquad\quad \} \\
\qquad \mathsf{in}\ \mathit{calpar}(\mathit{adds}(\mathit{makes}\ 28)(\mathit{makes}\ 32))
\end{array}
$$

Mivel p_1 típusa $\exists\,\alpha\,.\,A$, az unpack végrehajtásakor az x típusa $A\,[\,\alpha := \beta\,]$ lesz, ami éppen megfelel a let-ben szereplő típusnak, és így a let értékadásai végrahajtódnak, a let törzsében megadott kifejezés kiértékelődik.

Látható, hogy az unpack kifejezésben nem használjuk a p_1 csomagban elrejtett típus reprezentációját, a típusról csak annyit tudunk, hogy a típus a β. □

Az egzisztenciális típusról még megjegyezzük, hogy vannak egzisztenciális típusok, amelyek bár szintaktikusan helyesek, a gyakorlatban *önmagukban* nem használhatók. Ilyenek a

$$\exists\,\alpha\,.\,\alpha \to A$$

szerkezetű típusok, ahol A például egy alaptípus. Ha nincs olyan típus, ami az α típusváltozónak legalább egy lehetséges értékét magadja, akkor az α mindvégig ismeretlen marad.

Megemlítjük még a

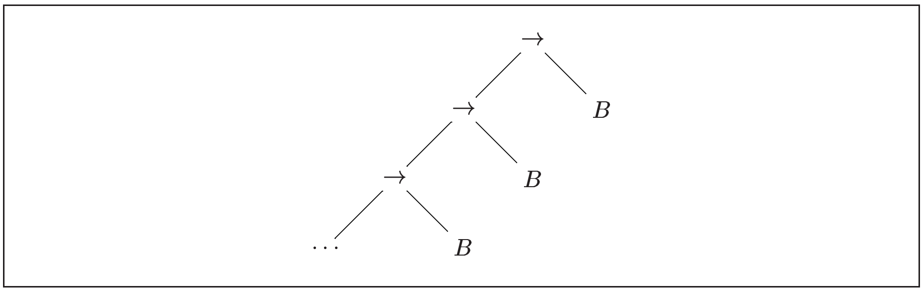
$$\exists\,\alpha\,.\,Pair_{A \,\to\, \alpha\times\alpha \,\to\, B}$$

típust. Látható, hogy ennek a típusnak a használatakor az α reprezentációjára egyáltalán nincs is szükség.

6.2. A rekurzív típus

A ??. példában láttuk, hogy az xx kifejezés nem típusozható. Ha a második x típusa A, akkor az első x típusának $A \to B$-nek kell lennie, és mivel ugyanarról az x-ről van szó, az $A = A \to B$ rekurzív egyenlőségnek teljesülnie kell.

Nyilvánvaló, hogy az A „rekurzív” típust csak végtelen szerkezettel tudjuk megadni, hiszen a leírás minden lépésében az A-t $A \to B$-vel kell helyettesíteni (6.1. ábra).



6.1. ábra. Az $A = A \to B$ típus

A λ-kifejezésekre vonatkozó $E = \ldots E \ldots$ rekurzív egyenlet megoldására az Y fixpont kombinátort vezettük be. Ehhez hasonlóan, a típusokra vonatkozó $A = \ldots A \ldots$ rekurzív egyenlet megoldására bevezetünk egy fixt „típusos" fixpont operátort.

Informálisan, ha az $A = A \to B$ egyenlőség jobboldalán levő A-ra végrehajtunk egy absztrakciót, és az absztrakció típusváltozóját α-val jelöljük, akkor egy $\forall\alpha\,.\,\alpha \to B$ típusú kifejezést kapunk. Ha erre az A típust applikáljuk, akkor

$$A = (\forall\alpha\,.\,\alpha \to B)[A].$$

Ebből az egyenletből látható, hogy A a $\forall\alpha\,.\,\alpha \to B$ kifejezés fixpontja. Ha a fixpontot a fixt operátor állítja elő, akkor

$$A = \mathsf{fixt}(\forall\alpha\,.\,\alpha \to B).$$

Bevezetve a $\mathsf{fixt}(\forall\alpha\,.\,C)$-re a $\mu\alpha\,.\,C$ jelölést,

$$A = \mu\alpha\,.\,\alpha \to B.$$

Általánosan tehát, ha a B típus kifejezésében szerepel az A típus, akkor a típusokra vonatkozó $A = B$ egyenlet megoldása $A = \mu\alpha\,.\,(B\,[\,A := \alpha\,])$, ahol $B\,[\,A := \alpha\,]$ azt jelenti, hogy a B-ben az A-kat α-val helyettesítjük.

Természetesen a fenti gondolatmenet nem formális levezetés, hiszen $\Lambda\alpha\,.\,A$ vagy $(\forall\alpha\,.\,A)\,[\,B\,]$ szerkezetű kifejezéseket még nem definiáltunk, a magyarázat csupán a μ operátor bevezetésének indoklására szolgál.

A μ konstruktorral képzett $\mu\alpha\,.\,A$ típust *rekurzív típusnak* nevezzük. A μ egy operátor, azaz típuskonstruktor, felhasználásával írható le a rekurzív típusegyenlet megoldása.

A $\mu\alpha\,.\,A$ típusban az α típusváltozó *hatásköre* az A típuskifejezés. Azt mondjuk, hogy a $\mu\alpha$ *köti* az A-ban levő szabad α típusváltozókat. A szokásos módon definiáljuk a $\mu\alpha\,.\,A\,[\,\alpha := B\,]$ *helyettesítést* is, a helyettesítés azt jelenti, hogy az A-ban a $\mu\alpha$ által kötött α típusváltozókat a B típuskifejezésre cseréljük.

A $\mu\alpha\,.\,A$ rekurzív típusban a típusváltozó megválasztása tetszőleges. Ha $\beta \notin FV(A)$, a $\mu\alpha\,.\,A$ típusból képezhetjük a $\mu\beta\,.\,A\,[\,\alpha := \beta\,]$ típust, és ezt a két típust nem tekintjük különbözőnek, azaz a típusok *ekvivalensek* lesznek. Ez a λ-kifejezések α-konverziójának felel meg.

Az F_2 típusrendszert a rekurzív típus bevezetésével bővítjük.

6.2.1. Definíció. A típusra vonatkozó szintaktikus szabályok:

$$\begin{array}{lcl}
\langle \textit{típus} \rangle & ::= & \langle \textit{típusváltozó} \rangle \\
 & \mid & (\,\langle \textit{típus} \rangle \to \langle \textit{típus} \rangle\,) \\
 & \mid & (\,\forall\,\langle \textit{típusváltozó} \rangle\ .\ \langle \textit{típus} \rangle\,) \\
 & \mid & (\,\mu\,\langle \textit{típusváltozó} \rangle\ .\ \langle \textit{típus} \rangle\,)
\end{array}$$

6.2.2. Példa. *(A $List_A$ típus)*
A rekurzív típus felhasználásával lehetőségünk van arra, hogy leírjuk az A típusú elemekből álló lista $List_A$ típusát.

Egy lista vagy nil, vagy cons x y, ahol nil$:List_A$, $x\!:\!A$ és $y\!:\!List_A$. Ha a cons típusát a *Pair* típussal írjuk le, a $List_A$ típus a következő *Union* típus lesz:

$List_A = Union_{List_A + Pair_{A \times List_A}}$.

A *Pair* típust a *szorzat* típussal, a *Union* típust az *összeg* típussal jelölve a

$List_A = List_A + (A \times List_A)$

típust kapjuk. Ebből a rekurzív egyenletből

$List_A \equiv \mu\alpha\,.\,\alpha + (A \times \alpha)$.

Ez valóban a rekurzív egyenlet megoldása, mivel

$List_A =$
$(\mu\alpha\,.\,\alpha + (A \times \alpha))\ [\,\alpha := \mu\alpha\,.\,\alpha + (A \times \alpha)\,] =$
$(\mu\alpha\,.\,\alpha + (A \times \alpha)) + (A \times (\mu\alpha\,.\,\alpha + (A \times \alpha)))) =$
$List_A + (A \times List_A)$. □

6.2.3. Példa. *(A NatTree típus)*
A levelein természetes számokat tartalmazó bináris fa a következőképpen adható meg. A fa vagy egy leaf levél, melyre leaf$:Nat$, vagy egy node x y csúcspont, ahol $x\!:\!NatTree$ és $y\!:\!NatTree$. Ebből a következő egyenlőséget kapjuk:

$NatTree = Nat + (NatTree \times NatTree)$,

amelyből a rekurzív típuskonstruktor felhasználásával

$NatTree \equiv \mu\alpha\,.\,Nat + (\alpha \times \alpha)$. □

6.2.4. Példa. *(Rekurzív rekord típus)*
Definiáljunk egy egyenesen mozgatható pontot a record kifejezés felhasználásával, a pont típusa legyen *Point*. A rekordnak legyen két mezője, *setx* és *getx*, az első a pont mozgatására, a második a pont helyzetének meghatározására szolgáljon. Ekkor

$Point \equiv \{\!|\, setx : Point \to Int \to Point,\ \ getx : Point \to Int \,|\!\}$

A *Point* rekurzív típusdefinícióját a μ operátor felhasználásával a következő rekurzív típusra alakíthatjuk át:

$Point \equiv \mu\alpha\,.\,\{\!|\, setx : \alpha \to Int \to \alpha,\ \ getx : \alpha \to Int \,|\!\}$ □

6.2.1. Rekurzív típusok ekvivalenciája

Az előző pontban láttuk, hogy a rekurzív típus típusváltozójának megválasztása tetszőleges, azaz érvényes egy, a λ-kifejezések α-konverziójához hasonló tulajdonság:

$\mu\alpha\,.\,A \equiv \mu\beta\,.\,A\,[\,\alpha := \beta\,], \quad \beta \notin FV(A)$.

Ebből a következő szabályt kaphatjuk meg:

$$\frac{\Gamma \vdash A\,[\,\alpha := \gamma\,] \equiv B\,[\,\beta := \gamma\,]}{\Gamma \vdash \mu\alpha\,.\,A \equiv \mu\beta\,.\,B} \quad [\textsc{Type Rec Equiv}]$$

A rekurzív típus használatakor észrevehetjük, hogy egy rekurzív típusnak több szintaktikusan helyes alakja is lehet. Ezt mutatjuk meg a következőkben.

6.2.5. Példa. *(Az Éhes függvény típusa)*
Legyen E egy olyan „*Éhes*” típusú függvény, azaz λ-absztrakció, amelyik minden argumentumát „megeszi”, de ettől nem változik meg. Az argumentumok legyenek természetes számok. Ekkor az

$E \equiv \lambda x : Nat\,.\,E$

kifejezésből az

$\textit{Éhes} = Nat \to \textit{Éhes}$

rekurzív típusegyenletet kapjuk, amelynek a megoldása

$\textit{Éhes} \equiv \mu\alpha\,.\,Nat \to \alpha$.

Megjegyezzük, hogy az E függvény az

$E \equiv \mathsf{Y}\ (\lambda y{:}Nat \to \textit{Éhes}\,.\,\lambda x{:}Nat\,.\,y)$

alakban írható fel. □

Tekintsük az előző példában szereplő E kifejezést, amelynek típusára az
$\acute{E}hes = Nat \to \acute{E}hes$
egyenlet megoldásával az
$\acute{E}hes = \mu\alpha \,.\, Nat \to \alpha$
típust kaptuk. Most vizsgáljuk meg a
$(Nat \to \alpha)[\alpha := \mu\alpha \,.\, Nat \to \alpha]$
kifejezést. A helyettesítést végrehajtva, majd a fenti azonosságokat felhasználva

$Nat \to (\mu\alpha \,.\, Nat \to \alpha) \equiv$
$Nat \to \acute{E}hes =$
$\acute{E}hes$
Hasonlóan, ezt az eredményt felhasználva a
$(Nat \to (Nat \to \alpha))[\alpha := \mu\alpha \,.\, Nat \to \alpha]$
helyettesítésből a
$Nat \to (Nat \to (\mu\alpha \,.\, Nat \to \alpha) \equiv$
$Nat \to (Nat \to \acute{E}hes) =$
$\acute{E}hes$
azonosságot kapjuk. Tehát az $\acute{E}hes$ típus alakjai:
$\acute{E}hes$
$\mu\alpha \,.\, Nat \to \alpha$
$Nat \to (\mu\alpha \,.\, Nat \to \alpha)$
$Nat \to (Nat \to (\mu\alpha \,.\, Nat \to \alpha))$
...
Láthatjuk, hogy egy típusnak több szintaktikusan helyes formája van. E típusok közötti kapcsolat leírására vezessük be a fold és az unfold függvényt. A típusrendszer kifejezéseinek szintaktikája ezekkel bővül:

6.2.6. Definíció. A kifejezésekre vonatkozó szintaktikus szabályok:

$$
\begin{array}{lcl}
\langle \lambda\textit{-kifejezés} \rangle & ::= & \langle \textit{változó} \rangle \\
 & | & (\,\lambda\,\langle \textit{változó} \rangle : \langle \textit{típus} \rangle\ .\ \langle \lambda\textit{-kifejezés} \rangle\,) \\
 & | & (\,\langle \lambda\textit{-kifejezés} \rangle\ \langle \lambda\textit{-kifejezés} \rangle\,) \\
 & | & (\,\Lambda\,\langle \textit{típusváltozó} \rangle\ .\ \langle \lambda\textit{-kifejezés} \rangle\,) \\
 & | & (\,\langle \lambda\textit{-kifejezés} \rangle\ [\ \langle \textit{típus} \rangle\ ]\,) \\
 & | & \mathsf{fold}\,[\ \langle \textit{típus} \rangle\ ]\ \langle \lambda\textit{-kifejezés} \rangle \\
 & | & \mathsf{unfold}\,[\ \langle \textit{típus} \rangle\ ]\ \langle \lambda\textit{-kifejezés} \rangle
\end{array}
$$

A $\mathsf{fold}\,[\,A\,]\,E$ és $\mathsf{unfold}\,[\,A\,]\,E$ kifejezéseket gyakran a $\mathsf{fold}_A\ E$ és $\mathsf{unfold}_A\ E$ alakban is írják.

A típusszabályok legyenek a következők:

6.2.7. Definíció. A rekurzív típusra vonatkozó szabályok:

$$\frac{\Gamma, \alpha \vdash A}{\Gamma \vdash \mu\alpha\,.\,A}\ [\textsc{Type Rec}]$$

$$\frac{\Gamma \vdash E{:}A\,[\,\alpha := \mu\alpha\,.\,A\,]}{\Gamma \vdash \mathsf{fold}\,[\,\mu\alpha\,.\,A\,]\,E\ :\ \mu\alpha\,.\,A}\ [\textsc{Val Fold}]$$

$$\frac{\Gamma \vdash E{:}\mu\alpha\,.\,A}{\Gamma \vdash \mathsf{unfold}\,[\,\mu\alpha\,.\,A\,]\,E\ :\ A\,[\,\alpha := \mu\alpha\,.\,A\,]}\ [\textsc{Val Unfold}]$$

A második és harmadik szabály azt is kimondja, hogy a fold és unfold operátorok első paraméterének rekurzív típusnak kell lennie.

$$
\begin{tikzcd}
\mu\alpha\,.\,A \arrow[rr, bend left, "{\mathsf{unfold}\,[\,\mu\alpha\,.\,A\,]}"] & & A\,[\,\alpha := \mu\alpha\,.\,A\,] \arrow[ll, bend left, "{\mathsf{fold}\,[\,\mu\alpha\,.\,A\,]}"]
\end{tikzcd}
$$

6.2. ábra. A fold és unfold kapcsolata

A fold és unfold műveletek operációs szemantikáját redukciókkal adjuk meg, ezeket a redukciókat is a $\twoheadrightarrow_\beta$ jellel jelöljük.

6.2.8. Lemma. A rekurzív típus operációs szemantikája:

$$\mathsf{unfold}\,[\,A\,]\,(\mathsf{fold}\,[\,B\,]\,E)\ \twoheadrightarrow_\beta\ E$$

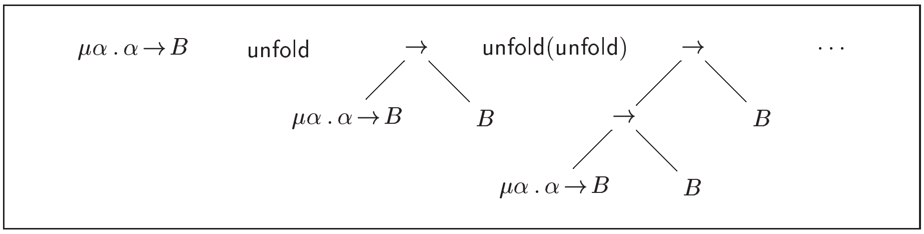
$$\mathsf{fold}\,[\,A\,]\,(\mathsf{unfold}\,[\,B\,]\,E)\ \twoheadrightarrow_\beta\ E$$

A fenti levezetésekben az unfold és a fold első argumentumának nem kell azonos típusnak lennie, mivel a redukció (mint minden redukció,) csak típusosan helyes kifejezésekre alkalmazható, és a típusellenőrzés biztosítja ezeknek a típusoknak az azonosságát.

A kérdés tehát az, hogy az unfold és a fold operátorokkal előállítható szintaktikusan különböző típusokat (6.3. ábra) azonosaknak tekintjük-e. Kétféle szemlélet alakult ki.

Az első ezeket a rekurzív típusokat *ekvivalens típusoknak* tekinti, és ekkor a rekurzív típust *equi-rekurzív típusnak* nevezzük. Az equi-rekurzív típusokat tartalmazó kifejezések típusellenőrzése nehéz feladat, mivel véges időben végtelen típus-szerkezetekkel kell foglalkozni.

A másik szemlélet egy rekurzív típus unfold és fold operátorokkal előállítható alakjait *izomorf típusoknak* tekinti, ekkor a rekurzív típust *izo-rekurzív típusnak* nevezzük. A különbözőeknek tekintett típus-formák közötti kapcsolatot az unfold és a fold függvények teremtik meg.



6.3. ábra. Az $\mathsf{unfold}\,[\,\mu\alpha\,.\,\alpha\to B\,]$ művelet

6.2.2. Az izo-rekurzív típusok alkalmazása

Látható, hogy az unfold és a fold operátorok a rájuk alkalmazott kifejezést nem változtatják meg, az operátorok csak a kifejezés típusát módosítják. Az operátoroknak szerepük a típusellenőrzésben, a típusok meghatározásában, típusosan helyes kifejezések megadásában van.

6.2.9. Példa. *(Az xx kifejezés típusa)*
A fejezet bevezetésében láttuk, hogy az xx kifejezésben az x típusa tetszőleges B-re $\mu\alpha\,.\,\alpha\to B$. Az xx applikáció típusosan akkor helyes, ha az első x típusa

$$\mathsf{unfold}\,[\,\mu\alpha\,.\,\alpha\to B\,] = (\mu\alpha\,.\,\alpha\to B)\to B.$$

Ebből azonnal látható, hogy

$(\mathsf{unfold}\,[\,\mu\alpha\,.\,\alpha\to B\,]\,x)x\ :\ B.$ □

Az izo-rekurzív típusok alkalmazásával olyan kifejezések is leírhatók típusosan helyes formában, amelyeknek nincs normál formájuk. A következő két példában az Ω kifejezését és az Y fixpont operátort írjuk le, és magadjuk a típusukat is.

6.2.10. Példa. *(Az $\Omega \equiv (\lambda x\,.\,xx)(\lambda x\,.\,xx)$ kifejezés)*
A $\lambda x\,.\,xx$ kifejezés típusosan helyes formáját E-vel jelölve, az előző példa eredményét felhasználva

$E \equiv \lambda x{:}\mu\alpha\,.\,\alpha\to B\,.\,((\mathsf{unfold}\,[\,\mu\alpha\,.\,\alpha\to B\,]\,x)\ x),$

és ennek a típusa

$E\ :\ (\mu\alpha\,.\,\alpha\to B)\to B.$

Ebből pedig már az Ω típusosan helyes kifejezése meghatározható,

$\Omega \equiv E(\mathsf{fold}\,[\,\mu\alpha\,.\,\alpha\to B\,]\,E).$

A B típusra egy típusabsztrakciót megadva

$\Omega\ :\ \forall\beta\,.\,\beta$

$$\begin{aligned}\Omega \equiv \Lambda\beta\,.\,&(\lambda x{:}\mu\alpha\,.\,\alpha\to\beta\,.\,((\mathsf{unfold}\,[\,\mu\alpha\,.\,\alpha\to\beta\,]\,x)\ x))\\ &(\mathsf{fold}\,[\,\mu\alpha\,.\,\alpha\to\beta\,]\,\lambda x{:}\mu\alpha\,.\,\alpha\to\beta\,.\,((\mathsf{unfold}\,[\,\mu\alpha\,.\,\alpha\to\beta\,]\,x)\ x)).\end{aligned}$$ □

Az Ω-nak a fenti példában szereplő kifejezése egy olyan típusosan helyes kifejezés, hogy tetszőleges A típusra $\Omega\,[\,A\,]\ \twoheadrightarrow_\beta\ \Omega{:}A$. Ennek felhasználásával bármikor meg tudunk adni egy tetszőleges A típusú kifejezést, hiszen $\Omega\,[\,A\,]$ éppen egy A típusú kifejezés lesz.

6.2.11. Példa. *(Az Y fixpont kombinátor)*
A rekurzív típus lehetőséget ad arra is, hogy az

$\mathsf{Y}\ \equiv \lambda x\,.\,(\lambda y\,.\,x(yy))(\lambda y\,.\,x(yy))$

fixpont kombinátor típusát is meghatározzuk. Az előző két példából már megismert módszereket alkalmazzuk. Az yy kifejezésből következik, hogy az y típusa tetszőleges B típusra $\mu\alpha\,.\,\alpha\to B$, és így a törzs részkifejezésére

$\lambda y{:}(\mu\alpha\,.\,\alpha\to B)\,.\,x((\mathsf{unfold}\,[\,\mu\alpha\,.\,\alpha\to B\,]\,y)y).$

Mivel $(\mathsf{unfold}\,[\,\mu\alpha\,.\,\alpha\to B\,]\,y)y\ :\ B$, az x típusa $B\to{?}$. Milyen típust célszerű a kérdőjel helyére írni? A fenti részkifejezés típusa

$(\mu\alpha\,.\,\alpha\to B)\to{?}.$

Mivel ezt a részkifejezést önmagára kell applikálni, a kifejezés típusának rekurzív típusnak kell lennie. Tehát a kérdőjel helyére a B típust kell írnunk, $x{:}B \to B$. Az előző példához hasonlóan, a B-re egy típusabsztrakcióval a következőket kapjuk:

$\mathsf{Y} \;\; : \forall\beta\,.\,(\beta\to\beta)\to\beta,$

$\mathsf{Y} \;\equiv \Lambda\beta\,.\,\lambda x{:}\beta\to\beta\,.\,(\lambda y{:}(\mu\alpha\,.\,\alpha\to\beta)\,.\,x((\mathsf{unfold}\,[\,\mu\alpha\,.\,\alpha\to\beta\,]\,y)y))$
$\qquad\qquad (\mathsf{fold}\,[\,\mu\alpha\,.\,\alpha\to\beta\,]\,\lambda y{:}(\mu\alpha\,.\,\alpha\to\beta)\,.\,x((\mathsf{unfold}\,[\,\mu\alpha\,.\,\alpha\to\beta\,]\,y)y))$

□

Az izo-rekurzív típusok lehetőséget adnak arra is, hogy típusosan helyes kifejezésekkel írjuk le a rekurzív típusokban levő típuskonstruktorokat is.

6.2.12. Példa. *(A nil_A és a cons kifejezés)*
A 6.2.2. példában láttuk, hogy az A típusú elemeket tartalmazó lista típusa

$List_A \equiv \mu\alpha\,.\,\alpha + (A\times\alpha).$

Ekkor tetszőleges $E{:}A$ kifejezésre

$\mathsf{nil}_A \;\; \equiv \mathsf{fold}\,[\,List_A\,]\,(\mathsf{inLeft}_{A+(A\times List_A)}\; E),$

$\mathsf{cons} \equiv \lambda x{:}A\,.\,\lambda y{:}List_A\,.\,\mathsf{fold}\,[\,List_A\,]\,(\mathsf{inRight}_{A+(A\times List_A)}\;\langle x,y\rangle\,).$

Könnyen belátható, hogy

$\mathsf{nil}_A \;\; : List_A,$

$\mathsf{cons} \; : A\to List_A\to List_A.$ □

A típusos λ-kalkulus Bevezetésében utaltunk rá, hogy a típus nélküli rendszerekben is van egy típus, és csak azért mondjuk, hogy típus nélküli a rendszer, mert típusokat nem tudunk megkülönböztetni, mindennek ez az egy típus a típusa. Most megmutatjuk, hogy a típus nélküli λ-kalkulusban mi ez az egy típus.

A típusnélküli λ-kalkulus minden kifejezéséhez hozzárendelünk egy típusos λ-kalkulus-beli kifejezést. A hozzárendelést jelöljük a $\rightsquigarrow$ jellel, az $E\in\Lambda$ kifejezés átalakítása legyen $E^{\rightsquigarrow}$.

6.2.13. Definíció. A $\rightsquigarrow$ konverzió algoritmusa:

> *A típusos rendszer egységes típusa legyen $A\equiv\mu\alpha\,.\,\alpha\to\alpha$. Az $E^{\rightsquigarrow}$ kifejezést az E szerkezete szerint definiáljuk:*
>
> $x \qquad \rightsquigarrow x$
>
> $\lambda x\,.\,E \rightsquigarrow \mathsf{fold}\,[\,A\,]\,\lambda x{:}A\,.\,E^{\rightsquigarrow}$
>
> $EF \quad\; \rightsquigarrow (\mathsf{unfold}\,[\,A\,]\,E^{\rightsquigarrow})F^{\rightsquigarrow}$

Az átalakítással kapott minden típusos kifejezés típusa A, és belátható, hogy ha a típusnélküli λ-kalkulusban $E \twoheadrightarrow^* F$, akkor az $E^{\rightsquigarrow} \twoheadrightarrow^* F^{\rightsquigarrow}$ is teljesül.

6.2.14. Példa. *(A $\lambda x\,.\,x$ kifejezés és típusa)*
$\lambda x\,.\,x \;\rightsquigarrow$

$\mathsf{fold}\,[\,A\,]\,\lambda x{:}A\,.\,\underline{x^{\rightsquigarrow}} \;\rightsquigarrow$

$\mathsf{fold}\,[\,A\,]\,\lambda x{:}A\,.\,x.$

A $\lambda x{:}A\,.\,x$ kifejezés típusa $A \to A \equiv (\mu\alpha\,.\,\alpha\to\alpha)\to(\mu\alpha\,.\,\alpha\to\alpha)$, ezt a $\mathsf{fold}\,[\,A\,]$ kifejezéshez applikálva az $A \equiv \mu\alpha\,.\,\alpha\to\alpha$ típust kapjuk. □

6.2.15. Példa. *(A $(\lambda x\,.\,x)y$ kifejezés és típusa)*
$(\lambda x\,.\,x)y \;\rightsquigarrow$

$(\mathsf{unfold}\,[\,A\,]\,(\lambda x\,.\,x)^{\rightsquigarrow})y^{\rightsquigarrow} \equiv$

$(\underline{\mathsf{unfold}\,[\,A\,]\,(\mathsf{fold}\,[\,A\,]\,\lambda x{:}A\,.\,x)})y^{\rightsquigarrow} \;\twoheadrightarrow_\beta$

$(\lambda x{:}A\,.\,x)\underline{y^{\rightsquigarrow}} \rightsquigarrow$

$(\lambda x{:}A\,.\,x)y.$

A típushelyességhez felhasználva azt, hogy $y : A$, a $(\lambda x : A\,.\,x)y$ kifejezés típusa $A \equiv \mu\alpha\,.\,\alpha\to\alpha.$ □

6.2.3. Rekurzív altípusok

A rekurzív típusokra az altípus fogalmat úgy kell definiálni, hogy az altípusokra a 3.7.. pontban megadott tulajdonságok érvényben maradjanak.

Nyilvánvalónak tűnik, hogy ha $A \le B$, akkor legyen $\mu\alpha\,.\,A \le \mu\alpha\,.\,B$. A rekurzív típusokkal a probléma az, hogy a rekurzív típus egy végtelen szerkezet leírása. Egy típus szerkezetének a következő elemét az unfold művelettel kapjuk meg, tehát a $\mu\alpha\,.\,A \le \mu\alpha\,.\,B$ fennállásához az

$\mathsf{unfold}^+\,[\,\mu\alpha\,.\,A\,] \le \mathsf{unfold}^+\,[\,\mu\alpha\,.\,B\,]$

relációkat kellene vizsgálnunk. Ez azonban nem jó módszer, mint azt a következő példa is mutatja.

6.2.16. Példa. *(Az unfold^+ problémája)*
Legyen

$A \equiv \mu\alpha\,.\,Nat\to\alpha,$

$B \equiv \mu\beta\,.\,Nat\to Nat\to\beta.$

Ekkor

$\mathsf{unfold}^1\,[\,A\,] \equiv$

$Nat\to(\mu\,.\,\alpha Nat\to\alpha),$

és

$\mathsf{unfold}^2\,[\,A\,] \equiv$
$\mathsf{unfold}\,[\,A\,]\,(\mathsf{unfold}\,[\,A\,]\,) \equiv$
$\mathsf{unfold}\,[\,A\,]\,(Nat\!\to(\mu\alpha\,.\,Nat\!\to\alpha)) \equiv$
$Nat\!\to(Nat\!\to(\mu\alpha\,.\,Nat\!\to\alpha)) \equiv$
$Nat\!\to Nat\!\to A.$

Ugyanakkor

$\mathsf{unfold}^1\,[\,B\,] \equiv$
$Nat\!\to Nat\!\to(\mu\beta\,.\,Nat\!\to Nat\!\to\beta) \equiv$
$Nat\!\to Nat\!\to B.$

Ha feltesszük, hogy $A \leq B$, akkor az $\mathsf{unfold}^2\,[\,A\,] \leq \mathsf{unfold}^1\,[\,B\,]$ egyenlőtlenségből a $Nat\!\to Nat\!\to A \leq Nat\!\to Nat\!\to B$ összefüggést kapjuk, amelyből a SUB ARROW szabály szerint az $A \leq B$ egyenlőtlenséget kapjuk vissza. □

A példából látható, hogy a rekurzív típusok altípusainak vizsgálatához nem használhatjuk az unfold véges iterációkat.

Most tegyük fel, hogy a rekurzív típusok altípus fogalmára érvényes a következő típusszabály:

$$\frac{\Gamma \vdash A \leq B}{\Gamma \vdash \mu\alpha\,.\,A \leq \mu\beta\,.\,B}$$

Sajnos a $\mu\alpha\,.\,A \leq \mu\beta\,.\,B$ reláció teljesüléséhez az $A \leq B$ feltétel kevés. Ezt mutatjuk meg a következő példában.

6.2.17. Példa. *(A $\mu\alpha\,.\,A \leq \mu\beta\,.\,B$ feltétel kevés)*
Tegyük fel, hogy $\alpha\!\to\!A \leq \beta\!\to\!B$, így a SUB ARROW szabály szerint $A \leq B$. A fenti szabály szerint a feltételből

$\mu\alpha\,.\,\alpha\!\to\!A \leq \mu\beta\,.\,\beta\!\to\!B$

következik. Mindkét oldalra egy unfold műveletet végrehajtva, a

$(\mu\alpha\,.\,\alpha\!\to\!A)\!\to\!A \leq (\mu\beta\,.\,\beta\!\to\!B)\!\to\!B$

egyenlőtlenségnek is fenn kell állnia. Ismét a SUB ARROW szabály szerint, ekkor

$\mu\alpha\,.\,\alpha\!\to\!A \geq \mu\beta\,.\,\beta\!\to\!B,$

azaz $A \geq B$, ami ellentmond a feltételnek. □

Megjegyezzük, hogy a rekurzív típusok fenti altípusfogalmának problémáját azok a típusok okozzák, amelyekben közvetlenül a $\to$ típuskonstruktor előtt van típus típusváltozója. A fenti példa típusai is ilyenek voltak.

A $\mu\alpha\,.\,A \leq \mu\beta\,.\,B$ reláció fennállásához az $\alpha \leq \beta$ egyenlőtlenséget is meg kell követelnünk, azaz ha $\alpha \leq \beta$ esetén $A \leq B$ teljesül, csak akkor mondhatjuk, hogy $\mu\alpha\,.\,A \leq \mu\beta\,.\,B$.

A típusszabályok közé tehát a következő szabályt kell felvenni:

6.2.18. Definíció. A rekurzív altípusra vonatkozó szabály:

$$\frac{\Gamma, \alpha \leq \beta \vdash A \leq B}{\Gamma \vdash \mu\alpha\,.\,A \leq \mu\beta\,.\,B} \quad [\textsc{Sub Rec}]$$

6.2.19. Példa. *(Listák közötti altípus)*
A 6.2.2. példa eredményét felhasználva a *Nat* típusú számokból álló lista

$List_{Nat} = List_{Nat} + (Nat \times List_{Nat})$

típusegyenletének megoldása $List_{Nat} \equiv \mu\alpha\,.\,\alpha + (Nat \times \alpha)$. Hasonlóan, az *Int* típusú elemeket tartalmazó

$List_{Int} = List_{Int} + (Int \times List_{Int})$

lista típusa $List_{Int} \equiv \mu\beta\,.\,\beta + (Int \times \beta)$.

A $List_{Nat} \leq List_{Int}$ reláció teljesüléséhez a rekurzív altípusra vonatkozó fenti szabály alapján be kell látni, hogy $\alpha \leq \beta$ esetén

$\alpha + (Nat \times \alpha) \leq \beta + (Int \times \beta)$.

Ez viszont teljesül, hiszen $\alpha \leq \beta$ esetén a Sub Pair szabály szerint $Nat \times \alpha \leq Int \times \beta$, és a Sub Union szabály alkalmazásával $\alpha + (Nat \times \alpha) \leq \beta + (Int \times \beta)$. □

A gyakorlatban két rekurzív típus közötti reláció eldöntéséhez nem elegendő a fenti típusszabályban leírt feltétel. Az izo-rekurzív típusok esetén a fold/unfold operátorok alkalmazásával egy típus egymástól különböző leírásait, azaz a típus végtelen szerkezetének egyes elemeit kapjuk meg. Emiatt tehát először a típusok ekvivelenciáját kell vizsgálni, és ha az ekvivalencia nem áll fenn, csak akkor lehet az altípus relációval foglalkozni.

6.2.20. Példa. *(Típusok ekvivalenciája és az altípus)*
Legyen

$List_{IntNat} \equiv \mu\beta\,.\,\beta + Int \times (\beta + (Nat \times \beta))$.

Mutassuk meg, hogy $List_{Nat} \leq List_{IntNat}$. A típusszabály szerint, ha $\alpha \leq \beta$, az

$\alpha + (Nat \times \alpha) \leq \beta + Int \times (\beta + (Nat \times \beta))$

relációt kell bizonyítani. Ez azonban nem lehetséges. De definiáljunk egy $List_{NatNat}$ listát a

$List_{NatNat} = List_{NatNat} + Nat \times (List_{NatNat} + (Nat \times List_{NatNat}))$

egyenlettel. A lista típusa

$List_{NatNat} \equiv \mu\gamma \,.\, \gamma + Nat \times (\gamma + (Nat \times \gamma))$.

Erre a típusra bizonyítható a $List_{Nat} \equiv List_{NatNat}$ ekvivalencia, és könnyen belátható a

$List_{NatNat} \leq List_{IntNat}$

reláció is. □

6.3. Függő típusok

Az elsőrendű típusos λ-kalkulusban a $\lambda x\!:\!A \,.\, E$ kifejezés argumentuma egy F kifejezés, azaz ebben a kalkulusban kifejezésekhez kifejezéseket rendelünk. Az F_2 másodrendű, polimorfikus λ-kalkulus $\Lambda\alpha \,.\, E$ kifejezésének argumentuma típus, a típusapplikáció eredménye egy kifejezés, tehát itt típushoz rendelünk kifejezést. Az F_ω kalkulusban a $\Lambda\alpha \,.\, A$ kifejezésnek egy B típus az argumentuma, az applikáció egy típust ad eredményül, ebben a kalkulusban típushoz típust rendelhetünk.

F_1	$\lambda x\!:\!A \,.\, E$	*kifejezés* $\Rrightarrow$ *kifejezés*
F_2	$\Lambda\alpha \,.\, E$	*típus* $\Rrightarrow$ *kifejezés*
F_ω	$\Lambda\alpha \,.\, A$	*típus* $\Rrightarrow$ *típus*

Látszik, hogy a fenti táblázatból a *kifejezés* $\Rrightarrow$ *típus* leképezés hiányzik. Ha a λ-kalkulust a *függő típusnak* nevezett típusfogalommal bővítjük, akkor a kalkulussal ez a leképezés is megvalósítható.

A $List_{Nat}$ típust, és a típusra vonatkozó nil és cons konstruktorokat, illetve a head, tail és empty függvényeket már ismerjük. Vezessünk be egy új típust, az n hosszúságú listák típusát, és ezt jelöljük $List^{(n)}_{Nat}$-nel.

Nyilvánvaló, hogy ebben az új lista-típusban $\mathsf{nil} : List^{(0)}_{Nat}$. Mivel cons konstruktor a lista hosszát eggyel növeli, a cons argumentumaként most azt is meg kell adni, hogy milyen hosszú listához fűzzük hozzá az új elemet. Legyen ez a hossz-argumentum a cons első argumentuma.

Erre az argumentumra egy absztrakciót kell megadnunk. Az így létrehozott függvény egy olyan műveletet ír le, amelyik az első argumentumából, az n-ből, azaz egy kifejezésből egy típust, a $List_{Nat}^{(n+1)}$ típust hozza létre.

Ezt az absztrakciót, megkülönböztetve a λ-kalkulus korábban már bevezetett absztrakcióitól, a Π jellel jelöljük, tehát az absztrakció $\Pi x\!:\!A\,.\,B$ alakú. Ezt a típust nevezzük *függő típusnak*. A Π *köti* az x változót, és az x változó *hatásköre* a B típus.

Így a cons konstruktor típusa függő típus:

$\mathsf{cons}\ :\ \Pi n\!:\!Nat\,.\,Nat \to List_{Nat}^{(n)} \to List_{Nat}^{(\mathsf{succ}\ n)}.$

Ha az F_1 típusrendszert a függő típus bevezetésével bővítjük, akkor az így kapott kalkulust *λP kalkulusnak* nevezzük.

6.3.1. Definíció. A λP kalkulus típusra vonatkozó szintaktikus szabályai:

$$
\begin{array}{lcl}
\langle \textit{típus} \rangle & ::= & \langle \textit{típusváltozó} \rangle \\
 & | & (\,\langle \textit{típus} \rangle \to \langle \textit{típus} \rangle\,) \\
 & | & (\,\Pi\,\langle \textit{változó} \rangle : \langle \textit{típus} \rangle\ .\ \langle \textit{típus} \rangle\,)
\end{array}
$$

A $\Pi x\!:\!A\,.\,B$ kifejezésnek akkor van szerepe, ha az x szerepel a B típusban, azaz ha valóban az x-től való függőségről van szó.

Ha a B típusban nincs x változó, azaz B nem függ x-től, akkor

$\Pi x\!:\!A\,.\,B \equiv A \to B,$

azaz látszik, hogy a függő típus absztrakciója a típusokra már az elsőrendű típusos λ-kalkulusban is szereplő függvénytípus általánosítása.

Ha a függő típust kiterjesztjük a típusváltozókra is, akkor a $\Pi\alpha\!:\!\star\,.\,B$ kifejezéssel a másodrendű típusos λ-kalkulus polimorfikus kifejezésének a típusát is le tudjuk írni. Ha $\alpha\!:\!A$, és $A\!:\!\square$, akkor

$\Pi\alpha\!:\!A\,.\,B \equiv \forall\alpha\,.\,B.$

6.3.2. Definíció. A λP kalkulus típusra vonatkozó szabályai:

$$\frac{\Gamma, x{:}A \vdash B}{\Gamma \vdash \Pi x{:}A\,.\,B} \quad [\text{Type Dep}]$$

$$\frac{\Gamma, x{:}A \vdash E{:}B}{\Gamma \vdash \lambda x{:}A\,.\,E \;:\; \Pi x{:}A\,.\,B} \quad [\text{Val Dep}]$$

$$\frac{\Gamma \vdash E{:}\Pi x{:}A\,.\,B \quad \Gamma \vdash F{:}A}{\Gamma \vdash EF \;:\; B\,[\,x := F\,]} \quad [\text{Val Dep Appl}]$$

A második szabály a függő típusú kifejezésekre vonatkozik. Látható, hogy a feltétel típuskörnyezetéből az $x{:}A$ itt is beépül az absztrakcióba.

A Val Dep Appl szabály szerint, ha $\lambda x : A.E \;:\; \Pi x : A.B$, a $(\lambda x : A.E)F$ applikáció akkor jól típusozott, ha az x és az F típusa azonos. Az applikáció típusa $B\,[\,x := F\,]$, azaz B-ben a szabad x változót F-fel kell helyettesíteni.

A $(\lambda x : A.E)F$ applikáció eredményét a szokásos β-redukcióval határozhatjuk meg,

$(\lambda x{:}A.E)F \;\twoheadrightarrow_\beta\; E[x := F]$.

A β-redukcióra vonatkozó típusszabály a következő:

6.3.3. Lemma. (A β-redukció típusszabálya)

Ha az $E[x := F]$-ben és a $B[x := F]$-ben az F szabad változói nem válnak az E, illetve a B kötött változóivá, akkor

$$\frac{\Gamma \vdash (\lambda x{:}A.E)F \;:\; B[x := F]}{\Gamma \vdash E[x := F] \;:\; B[x := F]} \quad [\text{Conv Dep } \beta]$$

6.3.4. Példa. *(A $List^{(n)}_{Nat}$ típus)*
Láttuk, hogy

$\mathsf{nil} \;:\; List^{(0)}_{Nat}$,

$\mathsf{cons} \;:\; \Pi n{:}Nat\,.\,Nat \to List^{(n)}_{Nat} \to List^{(\mathsf{succ}\ n)}_{Nat}$.

Ennek a típusnak az elemeire alkalmazható head és a tail függvények típusa a következő:

$\mathsf{head} \;:\; \Pi n{:}Nat\,.\,List^{(n)}_{Nat} \to Nat$

$\mathsf{tail} \;:\; \Pi n{:}Nat\,.\,List^{(n)}_{Nat} \to List^{(\mathsf{pred}\ n)}_{Nat}$ □

6.3.5. Példa. *(A $List^{(n)}_{Nat}$ típus)*
Legyen F egy olyan függvény, melyre

$F\,0 = [\,]$
$F\,1 = [\,1\,]$
$F\,2 = [\,1, 2\,]$
$F\,3 = [\,1, 2, 3\,]$
...

Az F függvény argumentuma egy Nat típusú szám, és az applikáció eredménye egy $List^{(n)}_{Nat}$ típusú lista. Tehát

$\mathsf{F} \ : \ \Pi x{:}Nat\,.\,List^{(x)}_{Nat}.$ □

6.3.6. Példa. *(n darab szám összeadása)*
Legyen $plusz(n)$ egy olyan függvény, amelyik n darab Nat típusú számot ad össze, az eredmény nyilvánvalóan egy Nat típusú szám lesz. Ha ezt a függvényt a plusz kifejezéssel adjuk meg, akkor

$\mathsf{plusz}\ 2 \ : \ Nat \to Nat \to Nat,$
$\mathsf{plusz}\ 3 \ : \ Nat \to Nat \to Nat \to Nat,$
...
$\mathsf{plusz}\ n \ : \ Nat^n \to Nat,$

tehát

$\mathsf{plusz} \ : \ \Pi x{:}Nat\,.\,Nat^x \to Nat.$ □

7. FEJEZET

Magasabb rendű típusrendszerek

A 3. fejezetben láttuk, hogy az F_1 rendszerben a típusok csak egy előre adott típushalmaz elemei lehettek, és a típusos λ-kifejezések csak ilyen típusokat tartalmazhattak. A 5. fejezetben bevezettük a típusváltozó fogalmát, és az F_2 rendszerben a polimorfikus függvények már lehetőséget adtak olyan kifejezések leírására, melyekből típus argumentumokkal típusos λ-kifejezést, azaz az F_1 rendszer kifejezéseit állíthattuk elő.

Arra azonban még nincs lehetőségünk, hogy típusból típust állítsunk elő, azaz hogy típuskonstruktorokat definiáljunk. Ezért az F_2 rendszert abban az irányban bővítjük, hogy olyan kifejezéseket is leírhassunk, amelyek egy F_2-beli típus argumentummal egy új F_2-beli típust adnak eredményül. Ez lesz az F_3 típusrendszer. Tovább bővítve a lehetőségeket, az F_4 típusrendszer típuskonstruktorai F_3-as függvénytípusú argumentumból képeznek új típusokat, azaz a F_3-beli argumentummal F_3-beli típusokat hoznak létre. Tovább haladva, az F_5 típusrendszer típuskonstruktorai F_4-es argumentumokkal F_4-es típusokat adnak eredményül. Így tehát egy $F_1, F_2, F_3, F_4, F_5, \ldots$ típusrendszer sorozatot kapunk, és a sorozat tagjainak az úniója lesz majd az F_ω típusrendszer.

7.1. Az F_3 típusrendszer

Először megadjuk az F_3 típusrendszer szintaxisát, majd a típusszabályokkal foglalkozunk, végül az F_3 operációs szemantikájának legfontosabb tulajdonságait tárgyaljuk.

7.1.1. Az F_3 rendszer szintaxisa

Az 5. fejezet bevezetésében láttuk, hogy a polimorfikus identitás függvény alakja

$\mathsf{id} \equiv \Lambda\alpha\,.\,\lambda x{:}\alpha\,.\,x,$

és ennek típusa $\forall\alpha\,.\,\alpha\to\alpha$. Ha meg akarjuk adni az A típus identitás függvényét, akkor ezt az $\mathsf{id}\,[\,A\,]$ applikációval tehetjük meg, és az applikáció eredményének típusa $A\to A$ lesz. Látható, hogy az id függvény minden egyes applikációjára *külön-külön* kell megadni az applikáció eredményének a típusát.

Ezt a problémát elkerülhetjük úgy, hogy bevezetünk egy olyan függvényt, aminek, az id függvényhez hasonlóan, egy A típus argumentuma lehet, és ez a függvény ezzel az argumentummal egy típust, az $\mathsf{id}\,[\,A\,]$ típusát adja eredményül. Ez a függvény tehát egy típusból típust állít elő, ezért ezt a függvényt ***típuskonstruktornak*** nevezzük. Legyen ez a függvény $\Lambda\alpha\,.\,\alpha\to\alpha$.

A $\Lambda\alpha\,.\,A$ kifejezésben az α típusváltozó ***hatásköre*** A, és itt is beszélhetünk ***kötött*** és ***szabad*** típusváltozókról, azt mondjuk, hogy a Λ típusabsztrakció ***köti*** az A-ban levő α szabad típusváltozókat.

A Λ-val jelölt absztrakció most is *jobbasszociatív*, az applikáció balasszociatív, és mint a λ-kifejezéseknél, az applikációnak nagyobb a *precedenciája*, mint az absztrakciónak.

A $\Lambda\alpha\,.\,A$ kifejezés egy ***típusabsztrakció***. Erre az absztrakcióra applikálható egy típus, az applikáció eredménye a β-redukciónak megfelelő művelettel határozható meg. A β-redukció végrehajtásának feltételei és a β-redukció művelete azonos a λ-kifejezéseknél megadottakkal, azzal a különbséggel, hogy itt a változók helyett típusváltozók, a kifejezések helyett típuskifejezések értendők.

Tehát a fenti $\Lambda\alpha\,.\,\alpha\to\alpha$ kifejezés egy olyan függvény, amely egy megfelelő argumentummal egy típust ad eredményül,

$$(\Lambda\alpha\,.\,\alpha\to\alpha)A \;\twoheadrightarrow_\beta\; A\to A.$$

A $\Lambda\alpha\,.\,\alpha\to\alpha$ függvény bevezetésével tehát már nem kell minden A típusra külön-külön megadnunk az $\mathsf{id}\,[\,A\,]$ típusát, a $\mathsf{id}\,[\,A\,]$ típusa $(\Lambda\alpha\,.\,\alpha\to\alpha)A$ lesz.

A típuskonstruktorban levő absztrakciót jelölhetjük ugyanazzal a Λ-val, mint amit a polimorfikus függvényeknél használunk, mivel ezek nem téveszthetők össze, hiszen a típuskonstruktor egy típus argumentummal egy típust, a polimorfikus függvény egy típus argumentummal egy λ-kifejezést ad eredményül.

Általánosan tehát, ha az E polimorfikus függvény típusa $\forall\alpha\,.\,A$, akkor az $E\,[\,B\,]$ applikáció eredménye egy kifejezés, míg $\Lambda\alpha\,.\,A$ egy függvény, melyre $(\Lambda\alpha\,.\,A)B$ egy típus, és $E\,[\,B\,]\;:\;(\Lambda\alpha\,.\,A)B$.

7.1.1. Példa. *(A* twice *függvény)*

A **??**. példában láttuk, hogy a twice függvény az F_2 rendszerben a

$$\Lambda\alpha\,.\,\lambda f{:}\alpha\to\alpha\,.\,\lambda x{:}\alpha\,.\,f(fx)$$

alakban írható le, és típusa

$\forall\alpha\,.\,(\alpha\to\alpha)\to\alpha\to\alpha.$

Ebből a twice függvényre a következő típuskonstruktort kapjuk:

$\Lambda\alpha\,.\,(\alpha\to\alpha)\to\alpha\to\alpha.$ □

Láttuk, hogy a $\Lambda\alpha\,.\,A$ kifejezés egy absztrakció, amelynek a változója α. Az F_1 és F_2 típusrendszerekben a λ-absztrakciók változóinak a típusát meg kellett adni, milyen típust lehet most hozzárendelni az α típusváltozóhoz? Az α tetszőleges típus lehet, azaz az összes jól típusozott kifejezés típusainak egyike. Tehát az α típusa a „típusok típusa”.

Jelöljük $\star$-gal az összes jól típusozott kifejezés típusát, és nevezzük ezt *fajtának*. A $\star$ tehát egy konstans fajta, ha az E kifejezés jól típusozott és $E:A$, akkor az A fajtája $\star$, amit az $A{:}\star$-gal jelölünk.

Felhasználva a fajta fogalmát, azt mondhatjuk, hogy a $\Lambda\alpha\,.\,A$ kifejezés változójának a típusa $\star$, azaz

$\Lambda\alpha{:}\star\,.\,A.$

Ha K egy fajta, akkor ez az absztrakció egy $\star\to K$ leképezést valósít meg.

Bővítsük az F_2 rendszert ezzel a fajta fogalommal és a fenti jelölésekkel, az így kialakított típusrendszert F_3 *rendszernek* vagy *harmadrendű típusos λ-kalkulusnak* nevezzük.

7.1.2. Definíció. Az F_3 rendszer szintaxisa:

$$
\begin{array}{lcl}
\langle \textit{fajta} \rangle & ::= & \star \\
 & | & (\,\star\to\langle \textit{fajta} \rangle\,) \\
\langle \textit{típus} \rangle & ::= & \langle \textit{típusváltozó} \rangle \\
 & | & (\,\langle \textit{típus} \rangle \to \langle \textit{típus} \rangle\,) \\
 & | & (\,\forall\,\langle \textit{típusváltozó} \rangle : \langle \textit{fajta} \rangle\;.\;\langle \textit{típus} \rangle\,) \\
 & | & (\,\Lambda\,\langle \textit{típusváltozó} \rangle : \langle \textit{fajta} \rangle\;.\;\langle \textit{típus} \rangle\,) \\
 & | & (\,\langle \textit{típus} \rangle\;\langle \textit{típus} \rangle\,) \\
\langle \lambda\textit{-kifejezés} \rangle & ::= & \langle \textit{változó} \rangle \\
 & | & (\,\lambda\,\langle \textit{változó} \rangle : \langle \textit{típus} \rangle\;.\;\langle \lambda\textit{-kifejezés} \rangle\,) \\
 & | & (\,\langle \lambda\textit{-kifejezés} \rangle\;\langle \lambda\textit{-kifejezés} \rangle\,) \\
 & | & (\,\Lambda\,\langle \textit{típusváltozó} \rangle : \langle \textit{fajta} \rangle\;.\;\langle \lambda\textit{-kifejezés} \rangle\,) \\
 & | & (\,\langle \lambda\textit{-kifejezés} \rangle\,[\,\langle \textit{típus} \rangle\,]\,)
\end{array}
$$

7.1.3. Példa. *(Az* id *függvény típuskonstruktora az F_3 rendszerben)*

Az id függvény alakja és típusa

$\mathsf{id} \equiv (\Lambda\alpha{:}\star\,.\,\lambda x{:}\alpha\,.\,x)\ :\ (\forall\alpha{:}\star\,.\,\alpha\to\alpha),$

és így a típuskonstruktora

$\Lambda\alpha{:}\star\,.\,\alpha\to\alpha,$

és a típuskonstruktor típusa, azaz fajtája

$\star\to\star.$ □

7.1.4. Példa. *(A* twice *függvény típuskonstruktora)*

A twice függvényre a következő típuskonstruktort kapjuk:

$\Lambda\alpha{:}\star\,.\,(\alpha\to\alpha)\to\alpha\to\alpha,$

és a fajtája

$\star\to\star.$ □

7.1.5. Példa. *(A $Pair_{\alpha\times\beta}$ típus)*

A *Pair* típus típuskonstruktora

$\Lambda\alpha{:}\star\,.\,\Lambda\beta{:}\star\,.\,Pair_{\alpha\times\beta}.$

A típuskonstruktor fajtája

$\star\to(\star\to\star),$

és ez a jobbasszociatívitás miatt

$\star\to\star\to\star$

alakban is írható. □

Megjegyezzük, hogy a fajtára megadott szintaktikus szabályok szerint az F_3 típusrendszerben a típuskonstruktorok fajtája csak „jobbról zárójelezett”, azaz

$\star,$

$\star\to\star,$

$\star\to(\star\to\star),$

$\star\to(\star\to(\star\to\star)),$

$\ldots$

felépítésű lehet. Ezekkel olyan többváltozós típuskonstruktorok fajtája írható le, amelyek valódi típusokat valódi típusra képeznek le.

7.1.2. Az F_3 típusrendszer szabályai

Az F_2 típusrendszer típuskörnyezete (7.4.2. definíció) az F_1-hez viszonyítva azzal bővült, hogy a típuskörnyezetbe bekerültek a típusváltozók. Az F_3 típusrendszer *típuskörnyezete* már a típusváltozók fajtáját, azaz az $\alpha : \langle \textit{fajta} \rangle$ párokat is tartalmazza.

7.1.6. Definíció. A típuskörnyezet szintaktikája:

$$
\begin{array}{lcl}
\langle \textit{típuskörnyezet} \rangle & ::= & \emptyset \\
& | & \langle \textit{típuskörnyezet} \rangle \,,\, \langle \textit{változó} \rangle : \langle \textit{típus} \rangle \\
& | & \langle \textit{típuskörnyezet} \rangle \,,\, \langle \textit{típusváltozó} \rangle \; : \langle \textit{fajta} \rangle
\end{array}
$$

A típuskörnyezetekre a 4.1.2. definícióban megadott feltételeket a típusváltozókra is meg kell adni, azaz

- ha α_i egy K_i fajtájú típusváltozó ($1 \le i \le n$), akkor egy típuskörnyezet az $\alpha_i\!:\!K_i$ párokból felépített

  $\alpha_1\!:\!K_1, \alpha_2\!:\!K_2, \ldots, \alpha_n\!:\!K_n$

  listát is tartalmazza,

- ha $i \neq j$, akkor $\alpha_i \neq \alpha_j$, azaz a típuskörnyezetben egy típusváltozó csak egyszer szerepelhet, ami azt jelenti, hogy egy típusváltozónak csak egy fajtája lehet.

Az F_3 típusrendszer *következtetéseiben* is megjelenik a fajta fogalma, mint egy jól formált fajta, és mint egy típus típusa.

7.1.7. Definíció. Az F_3 típusrendszer következtetései:

$\Gamma \vdash \mathit{wf}$	Γ jól formált környezet
$\Gamma \vdash A : K$	Γ-ban az A jól formált típus
$\Gamma \vdash E : A$	Γ-ban az E kifejezés típusa A
$\Gamma \vdash K$	Γ-ban a K jól formált fajta

7.1.8. Definíció. Az F_3 típusrendszer szabályai:

A környezetre vonatkozó szabályok:

$$\frac{}{\emptyset \vdash \mathit{wf}} \quad [\textsc{Env } \emptyset]$$

$$\frac{\Gamma \vdash A : \star \quad x \notin \mathit{dom}(\Gamma)}{\Gamma, x{:}A \vdash \mathit{wf}} \quad [\textsc{Env } x]$$

$$\frac{\Gamma \vdash K \quad \alpha \notin \mathit{dom}(\Gamma)}{\Gamma, \alpha : K \vdash \mathit{wf}} \quad [\textsc{Env } \alpha]$$

A fajtára vonatkozó szabályok:

$$\frac{\Gamma \vdash \mathit{wf}}{\Gamma \vdash \star} \quad [\textsc{Kind } \star]$$

$$\frac{\Gamma \vdash \star \quad \Gamma \vdash K}{\Gamma \vdash \star \to K} \quad [\textsc{Kind Arrow}]$$

$$\frac{\Gamma, \alpha{:}\star \vdash A : K}{\Gamma \vdash \Lambda\alpha{:}\star . A : \star \to K} \quad [\textsc{Kind Fun}]$$

$$\frac{\Gamma \vdash A : \star \to K \quad \Gamma \vdash B : \star}{\Gamma \vdash AB : K} \quad [\textsc{Kind Appl}]$$

A típusra vonatkozó szabályok:

$$\frac{\Gamma', \alpha : K, \Gamma'' \vdash \mathit{wf}}{\Gamma', \alpha : K, \Gamma'' \vdash \alpha : K} \quad [\textsc{Type } \alpha]$$

$$\frac{\Gamma \vdash A : \star \quad \Gamma \vdash B : \star}{\Gamma \vdash A \to B : \star} \quad [\textsc{Type Arrow}]$$

$$\frac{\Gamma, \alpha : K \vdash A : \star}{\Gamma \vdash \forall\alpha : K . A : \star} \quad [\textsc{Type Forall}]$$

Kifejezés típusára vonatkozó szabályok:

$$\frac{\Gamma', x{:}A, \Gamma'' \vdash \mathit{wf}}{\Gamma', x{:}A, \Gamma'' \vdash x : A} \quad [\textsc{Val } x]$$

$$\frac{\Gamma, x{:}A \vdash E : B \quad \Gamma \vdash A : \star}{\Gamma \vdash \lambda x{:}A . E : A \to B} \quad [\textsc{Val Fun}]$$

$$\frac{\Gamma \vdash E : A \to B \quad \Gamma \vdash F : A}{\Gamma \vdash EF : B} \quad [\textsc{Val Appl}]$$

$$\frac{\Gamma, \alpha : K \vdash E : A}{\Gamma \vdash \Lambda\alpha : K . E : \forall\alpha : K . A} \quad [\textsc{Val Fun2}]$$

$$\frac{\Gamma \vdash E : \forall\alpha : K . A \quad \Gamma \vdash B : K}{\Gamma \vdash E[B] : A[\alpha := B]} \quad [\textsc{Val Appl2}]$$

A Val Fun szabály feltételének második következtetése $\Gamma \vdash A : \star$. Ez azt jelenti, hogy az A típus jól formáltságát is vizsgálni kell. Mivel A egy absztrakció változójának a típusa, az A fajtája csak valódi típus lehet, nem lehet tetszőleges fajta. Természetesen, ha $\Gamma \vdash x : A$, akkor a $\Gamma \vdash A : \star$ következtetés is teljesül.

7.1.3. Az F_3 rendszer operációs szemantikája

Az F_2 rendszer oparációs szemantikája érvényes az F_3 rendszerre is, azzal a módosítással, hogy a típusváltozókra itt is meg kell adni a fajtákat.

Példaképpen csak a típuskonstruktorokra vonatkozó α, β és η-szabályokat és ezek típusszabályait adjuk meg.

7.1.9. Lemma. (Az α-konverzió típusszabálya)

$$\frac{\Gamma \vdash \Lambda\alpha{:}\star\,.\,A : \star\to L \quad \beta \notin FV(A) \quad \beta \notin dom(\Gamma)}{\Gamma \vdash \Lambda\beta{:}\star\,.\,A[\alpha := \beta] : \star\to L} \quad [\textsc{Conv}\ \alpha]$$

7.1.10. Lemma. (A β-redukció típusszabálya)

Ha az $A[\alpha := B]$-ben a B szabad típusváltozói nem válnak az A kötött típusváltozóivá, akkor

$$\frac{\Gamma \vdash (\Lambda\alpha{:}\star\,.\,A)B : L}{\Gamma \vdash A[\alpha := B] : L} \quad [\textsc{Conv}\ \beta]$$

7.1.11. Lemma. (Az η-konverzió típusszabálya)

Ha α az A-nak nem szabad változója, akkor

$$\frac{\Gamma \vdash \Lambda\alpha{:}\star\,.\,A\alpha : \star\to L}{\Gamma \vdash A : \star\to L} \quad [\textsc{Conv}\ \eta]$$

Az F_3 típusrendszerben a típuskonstruktorokra is teljesül a tárgyredukció és a normalizálás tétele is.

A β-redukcióra vonatkozó típusszabályból kapjuk a következő tételt, ami azt mondja ki, hogy a β-redukciók sorozata egy jól típusozott típuskonstruktor fajtáját nem változtatja meg.

7.1.12. Tétel. (A tárgyredukció tétele)

Ha $\Gamma \vdash A : K$ *és* $A \twoheadrightarrow^* B$, *akkor* $\Gamma \vdash B : K$.

Az F_3 típusrendszerben a típuskonstruktorokra is bevezethetjük a *normál forma* fogalmát.

7.1.13. Tétel. (A normalizálás tétele)

Az F_3 rendszerben minden jól típusozott típuskonstruktornak van normál formája.

Az F_3 típusrendszerben érvényes a Church-Rosser tétellel megfogalmazható rombusz-tulajdonság is.

7.2. Az $F_4, F_5, \ldots$ típusrendszerek

Az előző pontban láttuk, hogy a K fajta $\star$ vagy $\star \to K$ alakú. Ennek az volt a következménye, hogy a fajták csak „jobbról zárójelezett" kifejezések lehettek. A következő típusrendszerekben a fajták megadására további lehetőségeket adunk meg.

Vezessük be a *szint* fogalmát, és definiáljuk az egyes szintekhez tartozó fajtákat. Az i-edik szinthez tartozó fajták halmazát jelöljük $\mathcal{K}_i$-val. Legyen

$\mathcal{K}_0 = \emptyset,$

$\mathcal{K}_1 = \emptyset,$

$\mathcal{K}_2 = \{\star\},$

$\mathcal{K}_i = \{\star\} \cup \{K \to L \mid K \in \mathcal{K}_{i-1} \text{ és } L \in \mathcal{K}_i \quad (i \geq 3)\}.$

Tehát

$\mathcal{K}_0 = \emptyset$

$\mathcal{K}_1 = \emptyset$

$\mathcal{K}_2 = \{\star\}$

$\mathcal{K}_3 = \{\star,\ \star \to \star,\ \star \to (\star \to \star),\ \star \to (\star \to (\star \to \star)),\ \ldots\}$

$$\begin{aligned}\mathcal{K}_4 = \{&\star,\ \star \to \star,\ \star \to (\star \to \star),\ \star \to (\star \to (\star \to \star)),\ \ldots,\\ &(\star \to \star) \to \star,\ (\star \to \star) \to (\star \to \star),\ (\star \to \star) \to (\star \to (\star \to \star)),\ \ldots,\\ &(\star \to (\star \to \star)) \to \star,\ (\star \to (\star \to \star)) \to (\star \to \star),\\ &\qquad\qquad (\star \to (\star \to \star)) \to (\star \to (\star \to \star)),\ \ldots,\\ &\ldots\}\end{aligned}$$

Az F_1 típusrendszerben nincs típusváltozó, és így nincs típusabsztrakció sem. Az F_2 rendszerben már van típusváltozó, de még nincs típusabsztrakció, és a típusrendszerben definiálható típusok lehetséges fajtája a (nem jelölt) $\star = \mathcal{K}_2$. Az F_3-ban van típusváltozó és van típusabsztrakció is, a típusabsztrakció változójának fajtája $\star = \mathcal{K}_2$, és a típuskonstrukcióval a $\mathcal{K}_3$ elemei adhatók meg.

Ezzel analóg módon definiáljuk az F_i $(i \geq 1)$ ***típusrendszert*** úgy, hogy az F_i-ben a típuskonstruktor típusváltozójának fajtái legyenek a $\mathcal{K}_{i-1}$ elemei, és a típusabsztrakcióval létrehozható típusok fajtái legyenek a $\mathcal{K}_i$ halmaz elemei.

Az első öt típusrendszerben szereplő fajtákat a következő táblázat tartalmazza.

	típusabsztrakció típusváltozójának fajtája	típusok fajtái
F_1	$\mathcal{K}_0 = \emptyset$	$\mathcal{K}_1 = \emptyset$
F_2	$\mathcal{K}_1 = \emptyset$	$\mathcal{K}_2$
F_3	$\mathcal{K}_2$	$\mathcal{K}_3$
F_4	$\mathcal{K}_3$	$\mathcal{K}_4$
F_5	$\mathcal{K}_4$	$\mathcal{K}_5$

Látható, hogy az F_i indexének növekedésével a típusrendszerekben egyre bonyolultabb fajták megadása válik lehetővé. Megjegyezzük, hogy a funkcionális programok implementációjához általában már az F_3 típusrendszer is elegendő.

7.3. Az λ_ω típusrendszer

Mint az előző fejezetben láttuk, az F_ω típusrendszerhez az $F_2, F_3, F_4, \ldots$ típusrendszereken keresztül szeretnénk eljutni. Először az F_2 rendszerben megadtuk a típusabsztrakciót és típusapplikációt, majd egyre bonyolultabb típuskonstruktorok leírását tettük lehetővé.

Az F_ω típusrendszerhez el lehet jutni egy másik módszerrel is. Az F_1 rendszert első lépésben típuskonstruktorokkal bővítjük, ez lesz a λ_ω típusrendszer, és csak ezután vezetjük be az F_2-ben már megismert típusabsztrakciót és típusapplikációt.

Először röviden áttekintjük a λ_ω típusrendszert, és csak ezután foglalkozunk az F_ω-nak az $F_3, F_4, \ldots$ felöli megközelítésével.

A λ_ω rendszer tehát az F_1 rendszernek típuskonstruktorokkal, azaz típusabsztrakciókkal és típusapplikációkkal való bővítése. Az F_3 rendszerhez hasonlóan, a típusapplikációk típusos helyességéhez itt is be kell vezetni a fajta fogalmát. Ebben a szakaszban sötét színnel az F_1 rendszerhez viszonyított változásokat jelöljük.

7.3.1. Definíció. A λ_ω rendszer szintaxisa:

$$
\begin{array}{lcl}
\langle \textit{fajta} \rangle & ::= & \star \\
& | & (\,\langle \textit{fajta} \rangle \rightarrow \langle \textit{fajta} \rangle\,) \\
\langle \textit{típus} \rangle & ::= & \langle \textit{típusváltozó} \rangle \\
& | & (\,\langle \textit{típus} \rangle \rightarrow \langle \textit{típus} \rangle\,) \\
& | & (\,\Lambda\,\langle \textit{típusváltozó} \rangle : \langle \textit{fajta} \rangle\;.\;\langle \textit{típus} \rangle\,) \\
& | & (\,\langle \textit{típus} \rangle\;\langle \textit{típus} \rangle\,) \\
\langle \lambda\textit{-kifejezés} \rangle & ::= & \langle \textit{változó} \rangle \\
& | & (\,\lambda\,\langle \textit{változó} \rangle : \langle \textit{típus} \rangle\;.\;\langle \lambda\textit{-kifejezés} \rangle\,) \\
& | & (\,\langle \lambda\textit{-kifejezés} \rangle\;\langle \lambda\textit{-kifejezés} \rangle\,)
\end{array}
$$

Megjegyezzük, hogy a fajta itt, ellentétben az $F_3, F_4, \ldots$ típusrendszerekkel, tetszőleges $\langle \textit{fajta} \rangle \rightarrow \langle \textit{fajta} \rangle$ kifejezés lehet.

7.3.2. Példa. *(Az* id *függvény típuskonstruktora a* λ_ω *rendszerben)*
Az id függvény alakja

$\mathsf{id} \equiv \lambda x{:}\alpha\,.\,x$,

és így a típuskonstruktora

$\Lambda\alpha{:}\star\,.\,\alpha\rightarrow\alpha$,

és a típuskonstruktor típusa, azaz fajtája

$\star\rightarrow\star$. □

7.3.1. A λ_ω típusrendszer szabályai

A típuskörnyezet a típusváltozókkal és a fajtákkal bővül, természetesen a típusváltozó – fajta párokra a 7.3.1. alszakaszban leírt követelményeknek itt is teljesülniük kell.

7.3.3. Definíció. A típuskörnyezet szintaktikája:

$$
\begin{array}{lcl}
\langle \textit{típuskörnyezet} \rangle & ::= & \emptyset \\
& | & \langle \textit{típuskörnyezet} \rangle\,,\,\langle \textit{változó} \rangle : \langle \textit{típus} \rangle \\
& | & \langle \textit{típuskörnyezet} \rangle\,,\,\langle \textit{típusváltozó} \rangle : \langle \textit{fajta} \rangle
\end{array}
$$

A λ_ω típusrendszer *következtetéseiben* is megjelenik a fajta fogalma.

7.3.4. Definíció. A λ_ω típusrendszer következtetései:

$\Gamma \vdash wf$	Γ jól formált környezet
$\Gamma \vdash A : K$	Γ-ban az A jól formált típus
$\Gamma \vdash E : A$	Γ-ban az E kifejezés típusa A
$\Gamma \vdash K$	Γ-ban a K jól formált fajta

7.3.5. Definíció. A λ_ω típusrendszer szabályai:

A környezetre vonatkozó szabályok:

$$\frac{}{\emptyset \vdash wf} \quad [\textsc{Env}\ \emptyset]$$

$$\frac{\Gamma \vdash A\ :\star \quad x \notin dom(\Gamma)}{\Gamma, x{:}A \vdash wf} \quad [\textsc{Env}\ x]$$

$$\frac{\Gamma \vdash K \quad \alpha \notin dom(\Gamma)}{\Gamma, \alpha{:}K \vdash wf} \quad [\textsc{Env}\ \alpha]$$

A fajtára vonatkozó szabályok:

$$\frac{\Gamma \vdash wf}{\Gamma \vdash \star} \quad [\textsc{Kind}\ \star]$$

$$\frac{\Gamma \vdash K \quad \Gamma \vdash L}{\Gamma \vdash K \to L} \quad [\textsc{Kind Arrow}]$$

$$\frac{\Gamma, \alpha\ :\ K \vdash A\ :\ L}{\Gamma \vdash \Lambda\alpha{:}\star .\, A\ :\ K \to L} \quad [\textsc{Kind Fun}]$$

$$\frac{\Gamma \vdash A\ :\ K \to L \quad \Gamma \vdash B\ :\ K}{\Gamma \vdash AB\ :\ L} \quad [\textsc{Kind Appl}]$$

A típusra vonatkozó szabályok:

$$\frac{\Gamma', \alpha{:}K, \Gamma'' \vdash\ wf}{\Gamma', \alpha{:}K, \Gamma'' \vdash \alpha\ :\ K} \quad [\textsc{Type}\ \alpha]$$

$$\frac{\Gamma \vdash A\ :\star \quad \Gamma \vdash B\ :\star}{\Gamma \vdash A \to B\ :\star} \quad [\textsc{Type Arrow}]$$

Kifejezés típusára vonatkozó szabályok:

$$\frac{\Gamma', x{:}A, \Gamma'' \vdash wf}{\Gamma', x{:}A, \Gamma'' \vdash x\ :\ A} \quad [\textsc{Val}\ x]$$

$$\frac{\Gamma, x{:}A \vdash E\ :\ B \quad \Gamma \vdash A\ :\ \star}{\Gamma \vdash \lambda x{:}A .\, E\ :\ A \to B} \quad [\textsc{Val Fun}]$$

$$\frac{\Gamma \vdash E\ :\ A \to B \quad \Gamma \vdash F\ :\ A}{\Gamma \vdash EF\ :\ B} \quad [\textsc{Val Appl}]$$

A Val Fun szabály feltételének második következtetése, az F_3 típusrendszerhez hasonlóan, $\Gamma \vdash A : \star$, azaz az A típus jól formáltságát vizsgálni kell. Mivel A egy absztrakció változójának a típusa, az A fajtája most is csak valódi típus lehet.

7.4. Az F_ω típusrendszer

Az F_ω típusrendszert az $F_1, F_2, \ldots$ típusrendszerek egyesítéseként is definiálhatjuk, azaz legyen

$$F_\omega = \bigcup_{i \geq 1} F_i.$$

Az F_ω az $F_3, F_4, \ldots$ típusrendszerektől csak a fajta megadásában különbözik. Az F_ω-ban a fajta tetszőlegesen zárójelezett kifejezés lehet, formálisan tehát a fajta leírása a következő:

$$\begin{array}{lcl} \langle \mathit{fajta} \rangle & ::= & \star \\ & | & (\,\langle \mathit{fajta} \rangle \rightarrow \langle \mathit{fajta} \rangle\,) \end{array}$$

Mint ahogyan korábban is tettük, a felesleges zárójelpárokat a fajta leírásából most is elhagyhatjuk. A fajta leírása is *jobbasszociatív.*

Az F_ω kalkulusban tehát a fajta $\star$, vagy tetszőleges felépítésű $K \rightarrow L$ alakú kifejezés. Ha a *fajták típusát* $\square$-val jelöljük, akkor nyilvánvalóan

$$\star : \square,$$

és

$$K \rightarrow L : \square.$$

Ezekután megadjuk az F_ω rendszer szintaxisát, majd az F_ω típusrendszert.

7.4.1. Az F_ω rendszer szintaxisa

Az F_ω rendszer szintaxisának leírásában sötét színnel az F_3 rendszerhez viszonyított változásokat és bővítéseket jelöljük.

7.4.1. Definíció. Az F_ω rendszer szintaxisa:

$$
\begin{array}{lcl}
\langle \textit{fajták típusa} \rangle & ::= & \Box \\
\langle \textit{fajta} \rangle & ::= & \star \\
 & | & (\,\langle \textit{fajta} \rangle \rightarrow \langle \textit{fajta} \rangle\,) \\
\langle \textit{típus} \rangle & ::= & \langle \textit{típusváltozó} \rangle \\
 & | & (\,\langle \textit{típus} \rangle \rightarrow \langle \textit{típus} \rangle\,) \\
 & | & (\,\forall\,\langle \textit{típusváltozó} \rangle : \langle \textit{fajta} \rangle\ .\ \langle \textit{típus} \rangle\,) \\
 & | & (\,\Lambda\,\langle \textit{típusváltozó} \rangle : \langle \textit{fajta} \rangle\ .\ \langle \textit{típus} \rangle\,) \\
 & | & (\ \langle \textit{típus} \rangle\ \langle \textit{típus} \rangle\,) \\
\langle \lambda\textit{-kifejezés} \rangle & ::= & \langle \textit{változó} \rangle \\
 & | & (\,\lambda\,\langle \textit{változó} \rangle : \langle \textit{típus} \rangle\ .\ \langle \lambda\textit{-kifejezés} \rangle\,) \\
 & | & (\,\langle \lambda\textit{-kifejezés} \rangle\ \langle \lambda\textit{-kifejezés} \rangle\,) \\
 & | & (\,\Lambda\,\langle \textit{típusváltozó} \rangle : \langle \textit{fajta} \rangle\ .\ \langle \lambda\textit{-kifejezés} \rangle\,) \\
 & | & (\,\langle \lambda\textit{-kifejezés} \rangle\,[\,\langle \textit{típus} \rangle\,]\,)
\end{array}
$$

A definícióból látható, hogy a típusváltozók itt már tetszőleges fajtájú típusokat vehetnek fel értékként.

7.4.2. Az F_ω típusrendszer szabályai

A típuskörnyezet szintaktikája nem változik, megegyezik a korábbi F_3 típusrendszerrel, a következtetések formájában is csak annyi a változás, hogy a fajtának is meg kell adni a típusát.

7.4.2. Definíció. A típuskörnyezet szintaktikája:

$$
\begin{array}{lcl}
\langle \textit{típuskörnyezet} \rangle & ::= & \emptyset \\
 & | & \langle \textit{típuskörnyezet} \rangle\,,\,\langle \textit{változó} \rangle : \langle \textit{típus} \rangle \\
 & | & \langle \textit{típuskörnyezet} \rangle\,,\,\langle \textit{típusváltozó} \rangle\ :\ \langle \textit{fajta} \rangle
\end{array}
$$

7.4.3. Definíció. Az F_ω típusrendszer következtetései:

$\Gamma \vdash \mathit{wf}$	Γ jól formált környezet
$\Gamma \vdash A : K$	Γ-ban az A jól formált típus
$\Gamma \vdash E : A$	Γ-ban az E kifejezés típusa A
$\Gamma \vdash K : \Box$	Γ-ban a K jól formált fajta

7.4.4. Definíció. Az F_ω típusrendszer szabályai:

A környezetre vonatkozó szabályok:

$$\frac{}{\emptyset \vdash \mathit{wf}} \quad [\textsc{Env}\ \emptyset]$$

$$\frac{\Gamma \vdash A : \star \quad x \notin \mathit{dom}(\Gamma)}{\Gamma, x{:}A \vdash \mathit{wf}} \quad [\textsc{Env}\ x]$$

$$\frac{\Gamma \vdash K : \Box \quad \alpha \notin \mathit{dom}(\Gamma)}{\Gamma, \alpha{:}K \vdash \mathit{wf}} \quad [\textsc{Env}\ \alpha]$$

A fajtára vonatkozó szabályok:

$$\frac{\Gamma \vdash \mathit{wf}}{\Gamma \vdash \star : \Box} \quad [\textsc{Kind}\ \star]$$

$$\frac{\Gamma \vdash K : \Box \quad \Gamma \vdash L : \Box}{\Gamma \vdash K \to L : \Box} \quad [\textsc{Kind Arrow}]$$

$$\frac{\Gamma, \alpha : K \vdash A : L \quad \Gamma \vdash K : \Box}{\Gamma \vdash \Lambda\alpha{:}K\,.\,A : K \to L} \quad [\textsc{Kind Fun}]$$

$$\frac{\Gamma \vdash A{:} K \to L \quad \Gamma \vdash B : K}{\Gamma \vdash AB : L[\alpha := B]} \quad [\textsc{Kind Appl}]$$

A típusra vonatkozó szabályok:

$$\frac{\Gamma', \alpha{:}K, \Gamma'' \vdash \mathit{wf}}{\Gamma', \alpha{:}K, \Gamma'' \vdash \alpha : K} \quad [\textsc{Type}\ \alpha]$$

$$\frac{\Gamma \vdash A : \star \quad \Gamma \vdash B : \star}{\Gamma \vdash A \to B : \star} \quad [\textsc{Type Arrow}]$$

$$\frac{\Gamma, \alpha : K \vdash A : \star}{\Gamma \vdash \forall\alpha{:}K\,.\,A : \star} \quad [\textsc{Type Forall}]$$

Kifejezés típusára vonatkozó szabályok:

$$\frac{\Gamma', x : A, \Gamma'' \vdash \mathit{wf}}{\Gamma', x : A, \Gamma'' \vdash x : A} \quad [\textsc{Val}\ x]$$

$$\frac{\Gamma, x : A \vdash E : B \quad \Gamma \vdash A : \star}{\Gamma \vdash \lambda x{:}A\,.\,E : A \to B} \quad [\textsc{Val Fun}]$$

$$\frac{\Gamma \vdash E : A \to B \quad \Gamma \vdash F : A}{\Gamma \vdash EF : B} \quad [\textsc{Val Appl}]$$

$$\frac{\Gamma, \alpha : K \vdash E : A \quad \Gamma \vdash K : \Box}{\Gamma \vdash \Lambda\alpha{:}K\,.\,E : \forall\alpha{:}K\,.\,A} \quad [\textsc{Val Fun2}]$$

$$\frac{\Gamma \vdash E : \forall\alpha{:}K\,.\,A \quad \Gamma \vdash B : K}{\Gamma \vdash E[B] : A[\alpha := B]} \quad [\textsc{Val Appl2}]$$

7.4.3. Az F_ω típusrendszer operációs szemantikája

p 450, pp 553-...

7.5. A $\boldsymbol{\lambda}$-kocka

A 7.4. szakaszban bevezettük a fajták típusa fogalmat, amit $\Box$-val jelöltünk. A 7.4.1. definícióban láttuk, hogy a fajták típusának és a fajtának a szintaktikája a következőképpen írható le:

$$
\begin{array}{lcl}
\langle \textit{fajták típusa} \rangle & ::= & \Box \\
\langle \textit{fajta} \rangle & ::= & \star \\
 & | & (\langle \textit{fajta} \rangle \to \langle \textit{fajta} \rangle)
\end{array}
$$

Először a $K \to L$ fajtára egy absztrakciót definiálunk, majd ennek az absztrakciónak a jelöléseit kiterjesztjük a már korábban bevezetett absztrakciókra is.

7.5.1. A fajtaabsztrakció és a fajták típusa

Az F_ω kalkulusban a fajta $\star$, vagy tetszőleges felépítésű $K \to L$ alakú kifejezés lehet. A $K \to L$ fajta – mint az F_1 kalkulusban az $A \to B$ típus, – egy leképezést valósít meg, azaz egy olyan függvénynek felel meg, amelyik a K fajtát az L fajtára képezi le.

A $K \to L$ függvényre egy absztrakciót is megadhatunk, ezt az absztrakciót *fajtaabsztrakciónak* nevezzük. Jelöljük ezt az absztrakciót Π-vel. A függvény változója egy típusváltozó, amelynek a fajtája K, így a függvény alakja

$\Pi\alpha{:}K\,.\,L.$

Ekkor mondhatjuk, hogy az α típusváltozó *hatásköre* az L, és beszélhetünk az L-beli *szabad* és *kötött* típusváltozókról is.

Ha az L nem függ az α-tól, azaz az L-ben nincs szabad α típusváltozó, akkor

$\Pi\alpha{:}K\,.\,L \equiv K \to L.$

Ha a fajták típusát $\Box$-val jelöljük, akkor nyilvánvalóan

$\star\,:\,\Box,$

és

$(\Pi\alpha{:}K\,.\,L)\,:\,\Box.$

Egy fajtaabsztrakcióval és egy fajtával definiálhatjuk a *fajtaapplikációt* is. Az applikáció itt is az absztrakció törzsében az absztrakcióval kötött fajtaváltozóknak az argumentummal való helyettesítését jelenti.

Ezzel az fajtaabsztrakcióval a fajtának a 7.4.1. definícióban szereplő leírása így írható át:

7.5.1. Definíció. Az F_ω rendszer szintaxisa, a fajta:

$$\begin{array}{lcl} \langle \mathit{fajta} \rangle & ::= & \star \\ & | & (\,\Pi\alpha{:}\,\langle \mathit{fajta} \rangle\,.\,\langle \mathit{fajta} \rangle\,) \end{array}$$

A fajtaabsztrakció alakja nagyon hasonló a függő típus absztrakciójához, de ott az absztakció változója egy „λ-kifejezés változó”, míg a fajtaabsztrakció változója egy típusváltozó.

A fajtaabsztrakció bevezetésével a típuskörnyezet és a típusrendszer következtetéseinek formái nem változnak, csupán a típusrendszernek a fajtára vonatkozó szabályait kell átalakítanunk.

7.5.2. Definíció. Az F_ω típusrendszer szabályai, a fajta:

A fajtára vonatkozó szabályok:

$$\frac{\Gamma \vdash \mathit{wf}}{\Gamma \vdash \star : \square}\ [\textsc{Kind}\ \star]$$

$$\frac{\Gamma, \alpha{:}K \vdash L : \square}{\Gamma \vdash \Pi\alpha{:}K\,.\,L : \square}\ [\textsc{Kind Arrow}]$$

$$\frac{\Gamma, \alpha : K \vdash A : L \quad \Gamma \vdash K : \square}{\Gamma \vdash \Lambda\alpha{:}K\,.\,A : \Pi\alpha{:}K\,.\,L}\ [\textsc{Kind Fun}]$$

$$\frac{\Gamma \vdash A : \Pi\alpha{:}K\,.\,L \quad \Gamma \vdash B : K}{\Gamma \vdash AB : L[\alpha := B]}\ [\textsc{Kind Appl}]$$

7.5.2. Egységes típusszabályok

1. Vizsgáljuk meg az F_ω típusrendszer típusokat definiáló szabályait. Ezek a Kind Arrow, Type Arrow és a Type Forall szabályok.

$$\frac{\Gamma, \alpha{:}K \vdash L : \square}{\Gamma \vdash \Pi\alpha{:}K\,.\,L : \square}\ [\textsc{Kind Arrow}]$$

$$\frac{\Gamma, \alpha : K \vdash A : \star}{\Gamma \vdash \forall\alpha{:}K\,.\,A : \star}\ [\textsc{Type Forall}]$$

$$\frac{\Gamma \vdash A : \star \quad \Gamma \vdash B : \star}{\Gamma \vdash A \to B : \star} \quad [\textsc{Type Arrow}]$$

Írjuk át a második és harmadik szabályban az absztrakciókat, felhasználva a függő típusnál már megadott azonosságokat.

$\Pi\alpha{:}A\,.\,B \equiv \forall\alpha{:}A\,.\,B$,

$\Pi x{:}A\,.\,B \equiv A \to B$, ha a B típusban nincs x változó.

Tehát, a megfelelő módon átalakítva a feltételek következtetéseit is,

$$\frac{\Gamma \vdash K : \Box \quad \Gamma, \alpha{:}K \vdash L : \Box}{\Gamma \vdash \Pi\alpha{:}K\,.\,L : \Box} \quad [\textsc{Kind Arrow}]$$

$$\frac{\Gamma \vdash A : \Box \quad \Gamma, \alpha{:}A \vdash B : \star}{\Gamma \vdash \Pi\alpha{:}A\,.\,B : \star} \quad [\textsc{Type Forall}]$$

$$\frac{\Gamma \vdash A : \star \quad \Gamma, x{:}A \vdash B : \star}{\Gamma \vdash \Pi x{:}A\,.\,B : \star} \quad [\textsc{Type Arrow}]$$

Látható, hogy ha a változót és típusváltozót egyformán x-szel, a típusokat és a fajtákat egyformán A és B-vel jelöljük, akkor a fenti három szabály csak a következtetések típusaiban különbözik:

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash \Pi x{:}A\,.\,B : s_2} \quad [\textsc{Rule } s_1, s_2\,],$$

és az (s_1, s_2) értékei rendre $(\Box,\Box), (\Box,\star), (\star,\star)$.

2. Ehhez hasonlóan, az F_ω típusrendszerben több olyan szabálycsoportot találunk, amelyek összevonhatók. Vizsgáljuk most a Type α és Val x szabályokat.

$$\frac{\Gamma', \alpha{:}K, \Gamma'' \vdash \ \mathit{wf}}{\Gamma', \alpha{:}K, \Gamma'' \vdash \alpha : K} \quad [\textsc{Type } \alpha]$$

$$\frac{\Gamma', x{:}A, \Gamma'' \vdash \mathit{wf}}{\Gamma', x{:}A, \Gamma'' \vdash x : A} \quad [\textsc{Val } x]$$

Ha most is x-szel jelöljük a változót és típusváltozót, A-val a típust és a fajtát, akkor ez a két szabály csak a feltétel következtetésében levő típusban különbözik,

$$\frac{\Gamma', x{:}A, \Gamma'' \vdash \ A : s}{\Gamma', x{:}A, \Gamma'' \vdash x : A} \quad [\textsc{Rule Start}],$$

ahol az s értéke $\Box$, vagy $\star$.

3. A következő csoportot a Kind Fun, Val Fun és Val Fun2 szabályok alkotják.

$$\frac{\Gamma, \alpha : K \vdash A : L \quad \Gamma \vdash K : \Box}{\Gamma \vdash \Lambda\alpha{:}K\,.\,A : \Pi\alpha{:}K\,.\,L} \quad [\textsc{Kind Fun}]$$

$$\frac{\Gamma, x : A \vdash E : B \quad \Gamma \vdash A : \star}{\Gamma \vdash \lambda x{:}A\,.\,E : A \to B} \quad [\textsc{Val Fun}]$$

$$\frac{\Gamma, \alpha : K \vdash E : A \quad \Gamma \vdash K : \Box}{\Gamma \vdash \Lambda\alpha{:}K\,.\,E : \forall\alpha{:}K\,.\,A} \quad [\textsc{Val Fun2}]$$

Ha az $A \to B$-re és $\forall\alpha{:}K\,.\,A$-ra az Rule s_1, s_2 szabályban már használt $\Pi x{:}A\,.\,B : s$ függvényjelölést használjuk, akkor ezek a szabályok a

$$\frac{\Gamma, \alpha : K \vdash A : L \quad \Gamma \vdash K : \Box}{\Gamma \vdash \Lambda\alpha{:}K\,.\,A : \Pi\alpha{:}K\,.\,L} \quad [\textsc{Kind Fun}]$$

$$\frac{\Gamma, x : A \vdash E : B \quad \Gamma \vdash A : \star}{\Gamma \vdash \lambda x{:}A\,.\,E : \Pi\alpha{:}A\,.\,B} \quad [\textsc{Val Fun}]$$

$$\frac{\Gamma, \alpha : K \vdash E : A \quad \Gamma \vdash K : \Box}{\Gamma \vdash \Lambda\alpha{:}K\,.\,E : \Pi\alpha{:}K\,.\,A} \quad [\textsc{Val Fun2}]$$

alakba írhatók át. Ha ismét x jelöli a változókat, A, B a típust vagy a fajtát, E egy kifejezést vagy típuskifejezést, a λ és Λ absztrakciók jeleit egységesen λ-val jelöljük, akkor a három szabály közös alakja

$$\frac{\Gamma, x : A \vdash E : B \quad \Gamma \vdash A : s}{\Gamma \vdash \lambda x{:}A\,.\,E : \Pi x{:}A\,.\,B} \quad [\textsc{Rule Fun}],$$

ahol s értéke az első szabály esetén $\Box$, a második és harmadik szabály esetén $\star$.

4. Az F_ω típusrendszer három applikációs szabálya (Kind Appl, Val Appl és Val Appl2) is összevonható.

$$\frac{\Gamma \vdash A : K \to L \quad \Gamma \vdash B : K}{\Gamma \vdash AB : L[\alpha := B]} \quad [\textsc{Kind Appl}]$$

$$\frac{\Gamma \vdash E : A \to B \quad \Gamma \vdash F : A}{\Gamma \vdash EF : B} \quad [\textsc{Val Appl}]$$

$$\frac{\Gamma \vdash E : \forall\alpha{:}K\,.\,A \quad \Gamma \vdash B : K}{\Gamma \vdash E[B] : A[\alpha := B]} \quad [\textsc{Val Appl2}]$$

Ezekre a szabályokra megismételve a már többször alkalmazott átalakítási eljárásokat és jelöléseket, a következő szabályt kapjuk:

$$\frac{\Gamma \vdash E : \Pi x{:}A\,.\,B \quad \Gamma \vdash F : A}{\Gamma \vdash EF : B[x := F]} \quad [\text{Rule Appl}]$$

7.5.3. A λ-kocka típusrendszerei

Az előző alszakaszban láttuk, hogy az F_ω típusrendszer szabályait átalakítva, a fajtákra, típusokra, kifejezésekre a következő szabályrendszert kapjuk:

7.5.3. Definíció. Az F_ω típusrendszer egységes szabályai:

$$\frac{\Gamma \vdash wf}{\Gamma \vdash \star : \Box} \quad [\text{Kind } \star]$$

$$\frac{\Gamma', x{:}A, \Gamma'' \vdash A : s}{\Gamma', x{:}A, \Gamma'' \vdash x : A} \quad [\text{Rule Start}],$$

$$\frac{\Gamma, x : A \vdash E : B \quad \Gamma \vdash A : s}{\Gamma \vdash \lambda x{:}A\,.\,E : \Pi x{:}A\,.\,B} \quad [\text{Rule Fun}],$$

$$\frac{\Gamma \vdash E : \Pi x{:}A\,.\,B \quad \Gamma \vdash F : A}{\Gamma \vdash EF : B[x := F]} \quad [\text{Rule Appl}]$$

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash \Pi x{:}A\,.\,B : s_2} \quad [\text{Rule } s_1, s_2\ ],$$

ahol

$s \in \{\Box, \star\}$,

$(s_1, s_2) \in \{(\Box,\Box), (\Box,\star), (\star,\star)\}$.

Látható, hogy ha a Rule s_1, s_2 szabályban $(s_1, s_2) \in (\star,\star)$, akkor éppen az F_1 típusrendszer szabályait kapjuk meg. Az is azonnal nyilvánvaló, hogy ha $(s_1, s_2) \in \{(\Box,\star), (\star,\star)\}$, akkor a fenti szabályok az F_2, azaz a másodrendű polimorfikus típusrendszer szabályait állítják elő, és $(s_1, s_2) \in \{(\star,\Box), (\star,\star)\}$ esetén a függő típus szabályaihoz jutunk.

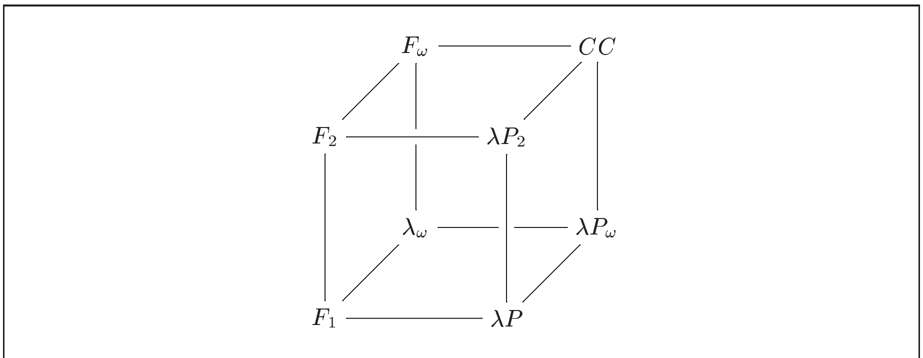
Az (s_1, s_2) lehetséges értékei $\{(\Box,\Box), (\Box,\star), (\star,\Box), (\star,\star)\}$, amelyek a következő függőségeket írják le:

	függőség	*példa*
$(\Box,\Box)$	típus függ a típus argumentumtól	$\Lambda\alpha{:}\star\,.\,\alpha \to \alpha : \star \to \star$
$(\Box,\star)$	kifejezés függ a típus argumentumtól	$\Lambda\alpha\,.\,\lambda x{:}\alpha\,.\,x : \forall\alpha\,.\,\alpha \to \alpha$
$(\star,\Box)$	típus függ a kifejezés argumentumtól	$\Pi x{:}A\,.\,B : \Box$
$(\star,\star)$	kifejezés függ a kifejezés argumentumtól	$\lambda x{:}A\,.\,x : A \to A$

Az (s_1, s_2) megválasztására nyolc lehetőségünk van. A fenti típusrendszerrel tehát nyolc különböző típusrendszert tudunk leírni. Ezeket a következő táblázatban adjuk meg.

típusrendszer	(s_1, s_2)			
F_1	$(\star, \star)$			
F_2	$(\star, \star)$	$(\square, \star)$		
λP	$(\star, \star)$		$(\star, \square)$	
λP_2	$(\star, \star)$	$(\square, \star)$	$(\star, \square)$	
λ_ω	$(\star, \star)$			$(\square, \square)$
F_ω	$(\star, \star)$	$(\square, \star)$		$(\square, \square)$
λP_ω	$(\star, \star)$		$(\star, \square)$	$(\square, \square)$
CC	$(\star, \star)$	$(\square, \star)$	$(\star, \square)$	$(\square, \square)$

A fenti típusrendszerek hierachiája a Barendregt-féle λ-kockával ábrázolható (7.1. ábra).



7.1. ábra. A λ-kocka

Látható, hogy a kocka jobb oldalára a függő típusok a jellemzőek, míg a funkcionális programozásban szerepet játszó kalkulusok a kocka bal oldalán találhatók meg.

A λP_2 a polimorfikus függő típusos kalkulust jelöli, a λP_ω kalkulusban a fajta kerül bevezetésre. CC a ***konstrukciók kalkulusát*** jelenti, amelynek gyakorlati alkalmazásai a tételbizonyító rendszerek megalapozásaiban találhatók meg. Ezekkel a témakörökkel ebben a könyvben nem foglalkozunk.

7.6. Az $F^{\omega}_{\leq}$ típusrendszer

Bővítés: Altípus, egzisztenciális típus, rekord típus. Ez majd az OO-hoz kell.

8. FEJEZET

Curry-Howard izomorfizmus

8.1. Az intuícionista minimális logika

Ebben a szakaszban az *ítéletlogika* egy olyan egyszerűsített változatát adjuk meg, amelyben csak egy *logikai összekötőjel*, az *implikáció* van értelmezve. Ezt az ítéletlogikát fogjuk *minimális logikának* nevezni. Ha az intuícionista ítéletlogikából indulunk ki, akkor az *intuícionista* minimális logikát kapjuk meg. Az intuícionista minimális logikát $IPC_{\rightarrow}$-lal jelöljük.

A klasszikus logika központi fogalma az *igazságérték*, az *igaz* vagy *hamis* igazságérték „abszolút" olyan értelemben, hogy egy állítás igazságértéke független minden más körülménytől, cselekménytől, hatástól, még akkor is, ha ezek az állításhoz közvetlenül hozzákapcsolódnak. Tehát egy állítás igaz vagy hamis, a „nem igaz" a hamis, és fordítva, a „nem hamis" az igaz. Harmadik eset nincs, azaz a *tertium non datur* elve szerint $p \vee \neg p$ minden p-re fennáll.

Azonban nem minden állításról lehet eldönteni, hogy igaz vagy hamis. A példa erre a következő közismert tétel:

Van két olyan x és y irracionális szám, melyre x^y racionális.

Az állítás bizonyítása egyszerű. Ha $\sqrt{2}^{\sqrt{2}}$ racionális szám, akkor legyen $x = y = \sqrt{2}$, ellenkező esetben legyen $x = \sqrt{2}^{\sqrt{2}}$ és $y = \sqrt{2}$.

A probléma ezzel a bizonyítással az, hogy nem tudjuk, a két lehetőség közül melyik igaz. Ehhez kevés az információ a bizonyításban, a bizonyítás nem *konstruktív*.

A konstruktív matematika a „létezik" fogalomra sokkal nagyobb hangsúlyt helyez, mint a klasszikus matematika. A konstruktív matematikában egy objektum létezése azt jelenti, hogy megadható egy olyan módszer, amely megkonstruálja az objektumot.

Egy természetes szám konstrukciója a szám megadását, vagy egy olyan eljárás megadását jelenti, amelyik ezt a számot előállítja. Egy (végtelen) sorozat konstrukciója azt jelenti, hogy van olyan függvény, amelyik minden n indexre előállítja a sorozat a_n elemét. Egy $f : A \to B$ függvény konstrukciója egy olyan algoritmus megadása, amelyik az A minden a eleméhez megadja a B egy $f(a)$ elemét. Az algoritmusok megadásánál az esetszétválasztás okozhat problémát, például a következő függvény megadásában:

$$f(n) = \begin{cases} 1, & \textit{ha a } \pi \textit{ szám decimális leírásában van } n \textit{ darab egymásutáni 7-es} \\ & \textit{számjegy,} \\ 0, & \textit{egyébként.} \end{cases}$$

Nyilvánvaló, hogy nem tudjuk leírni véges számjegysorozattal a π értékét, és nem tudunk olyan algoritmust adni, amelyik eldönti, hogy van-e ilyen 7-es sorozat. Ha a π leírásának k hosszúságú kezdő leírásában nincs, akkor lehet, hogy ez azért van, mert a π leírásában valóban nincs ilyen sorozat, vagy az is lehet, hogy csak kevés számjegyet adtunk meg.

A konstruktív matematikában a függvények leírására esetszétválasztást tehát csak akkor használhatunk, ha az esetek feltételeire eredményes döntési algoritmusokat tudunk adni.

Ezek a példák is mutatják a klasszikus logika problémáját, és azt, hogy miért volt szükség a konstruktív, vagy másnéven az *intuícionista logika* megalkotására. Az intuícionista logika alapjait Brouwer holland matematikus dolgozta ki az 1920-as években.

Az intuícionista logika nyelve azonos a klasszikus logika nyelvével. A szabályok is azonosak, kivéve a $\neg\neg A \to A$ axiómát, amelyet az intuícionista logikában az $A \to \neg A \to B$ axiómára kell kicserélni. A két logika közötti különbség szemantikai: a logikai összekötőjelek és a kvantorok interpretálásának módja más.

Az intuícionista logikában egy formula igaz, ha az igazsága valamilyen módon, például levezetéssel, bizonyítással „tanúsítható". A hangsúly az igazság fogalmáról a *konstrukció* fogalmára kerül át, a következtetések az állítások bizonyításának, megkonstruálásának egzisztenciáján alapulnak.

8.1.1. Definíció. A minimális logika formulái:

Jelöljük a minimális logika formuláit az $A, B, C, \ldots$ betűkkel. Ha A és B két formula, akkor $(A \to B)$ is a minimális logika formulája. $A \to$ az implikáció jele.

A formulák egyszerűbb leírása érdekében a legkülső zárójelpárt elhagyhatjuk, és feltesszük azt, hogy az implikáció *jobbasszociatív*.

8.1.2. Példa. *(Formulák)*
A következő kifejezések a minimális logika formulái:

A,

$A \to B$

$A \to B \to C \equiv A \to (B \to C)$ □

Egy A állítás bizonyítását, konstrukcióját az A állítás szerkezete szerint adjuk meg. A szabályokat *BHK-interpretációnak* is nevezik Brouwer, Heyting és Kolmogorov nevének kezdőbetűje alapján.

8.1.3. Definíció. Az implikáció BHK-interpretációja:

A minimális logika összekötőjelének, az implikációnak a konstruktív szemantikája a következő.

- *Az $A \to B$ konstrukciója egy olyan eljárás (függvény), amelyik az A minden konstrukcióját a B konstrukciójába transzformálja.*

8.1.4. Példa. *(Az $A \to A$ konstrukciója)*
Az $f \equiv \lambda x \,.\, x$ függvény az $A \to A$ állítás konstrukciója, hiszen ha a az A egy konstrukciója, akkor $fa = (\lambda x \,.\, x)a = a$, és így nyilvánvalóan az eredmény is az A konstrukciója. □

8.1.5. Példa. *(Az $A \to B \to A$ konstrukciója)*
Az előző példa gondolatmenetével, az $A \to B \to A$ állítás konstrukciója a $\lambda xy \,.\, x$ függvény. □

Ha definiáljuk a $\perp$ abszurdumot, azaz a konstans *false* értékű formulát, akkor a minimális logikában a negáció logikai összekötőjel is értelmezhető. A $\perp$ BHK-interpretációja:

- Jelölje $\perp$ az abszurdum formulát. A $\perp$-hoz nincs konstrukció.

8.1.6. Példa. *(A $\perp \to A$ konstrukciója)*
A $\perp \to A$ állítás konstrukciója tetszőleges függvény lehet, például legyen a $\lambda x \,.\, 1$ függvény. Vizsgálnunk kell a $\perp$ konstrukcióinak transzformáltjait. Legyen a a $\perp$ egy konstrukciója, de a BHK-interpretáció szerint a $\perp$-nak nincs konstrukciója, ellentmondásra jutottunk, ezért bármi, speciálisan az 1 értékű konstans függvény is az A konstrukciója. □

A $\neg A$ nem más, mint az $A \to \perp$ implikáció rövid leírása, azaz $\neg A$ azt jelenti, hogy az A az abszurdumhoz, a képtelenséghez vezet. Ebből az következik, hogy $\neg A$ konstrukciója egy olyan eljárás, amely az A minden konstrukcióját egy nemlétező objektumba képezi le.

8.1.7. Példa. *(A kontrapozíció törvénye)*
A kontrapozíció $(A \to B) \to (\neg B \to \neg A)$ BHK-interpretációja a következő. Legyen f az $A \to B$ konstrukciója, és meg kell határoznunk a $\neg B \to \neg A$ konstrukcióját. Tegyük fel, hogy $\neg B$ konstrukciója g, és A konstrukciója x. Ekkor fx a B-t állítja elő, és így $g(fx)$ eredménye a $\perp$ abszurdum. A $\lambda x\,.\,g(fx)$ függvény a $\neg A$-t adja, és így $\lambda g\,.\,\lambda x\,.\,g(fx)$ a $\neg B \to \neg A$ függvénye. Tehát a kontrapozíció konstrukciója

$\lambda f\,.\,\lambda g\,.\,\lambda x\,.\,g(fx).$ □

8.1.1. Természetes levezetés

A minimális logikában a szemantikus eldöntésprobléma megoldására a *természetes levezetés* technikáját alkalmazzuk.

A természetes levezetéssel megadott bizonyításban állítások szerepelnek, az állításoknak egy vagy több feltételük lehet. Azt mondjuk, hogy egy állítás bizonyított, ha az állításnak már nincs feltétele.

Technikai okokból a feltételeket megcímkézzük, ez majd arra lesz jó, hogy a levezetésben szereplő azonos feltételeket megkülönböztethessük. Legyen a címkék halmaza $\mathcal{V}$. Ha x egy címke, azaz $x \in \mathcal{V}$ és A egy formula, akkor $x{:}A$ jelölje azt, hogy az A formulát az x címkével jelöltük meg. Legyen Γ az ilyen megcímkézett feltételek halmaza.

Az intuícionista logikában a következtetésekhez tartozó konstrukciókat is meg kell adni. A konstrukciókat az F_1, azaz az elsőrendű típusos λ-kalkulus λ-kifejezéseivel írjuk le. A $\Gamma \vdash_E A$ jelölést használjuk annak a leírására, hogy a Γ (megcímkézett) feltételek következménye az A formula, és a következtetéshez tartozó bizonyítás, azaz konstrukció az E. Például, ha $x{:}A, y{:}B \vdash_x A$, akkor x az A formula konstrukciója.

Az implikáció BHK-interpretációja tehát úgy fogalmazható meg, hogy az $A \to B$ implikáció konstrukciója egy $(\lambda x{:}A\,.\,E){:}A \to B$ kifejezés.

Mivel a minimális logikában csak egy logikai összekötőjel van, az azonosság törvényén kívül csak az implikációra kell megadnunk a logikai szabályokat. Az intuícionista minimális logika nyelve megegyezik a minimális logika nyelvével, és a következtetési rendszerük is azonos.

Az $IPC_{\to}$ természetes levezetésének bizonyító rendszere egy axiómával, és három levezetési szabállyal adható meg.

8.1.8. Definíció. Az intuícionista minimális logika természetes levezetésének szabályai:

$$\frac{}{\Gamma, x{:}A \vdash_x A} \ [\textsc{Ident Axiom}]$$

$$\frac{\Gamma \vdash_E B}{\Gamma, x{:}A \vdash_E B} \ [\textsc{Assump}\ A]$$

$$\frac{\Gamma, x{:}A \vdash_E B}{\Gamma \vdash_{\lambda x:A\,.\,E} A \to B} \ [\textsc{Impl Intro}]$$

$$\frac{\Gamma \vdash_E A \to B \quad \Gamma \vdash_F A}{\Gamma \vdash_{EF} B} \ [\textsc{Impl Elim}]$$

A fentiek szerint tehát egy A formula bizonyítható, ha $\vdash_E A$, és a leírásból az is látszik, hogy az A-hoz tartozó konstrukció az E.

8.1.9. Példa. *(Az $A \to A$ formula levezetése)*

$$\frac{\frac{}{x{:}A \vdash_x A}}{\vdash_{\lambda x:A\,.\,x} A \to A}$$ □

8.1.10. Példa. *(Az $A \to B \to A$ formula levezetése)*

$$\frac{\frac{\frac{\frac{}{x{:}A \vdash_x A}}{x{:}A, y{:}B \vdash_x A}}{x{:}A \vdash_{\lambda y:B\,.\,x} B \to A}}{\vdash_{\lambda x:A\,.\,\lambda y:B\,.\,x} A \to (B \to A)}$$ □

8.1.11. Példa. *(Az $A \to A \to A$ formulának két levezetése van)*

$$\frac{\frac{\frac{\frac{}{x{:}A \vdash_x A}}{x{:}A, y{:}A \vdash_x A}}{x{:}A \vdash_{\lambda y:A\,.\,x} A \to A}}{\vdash_{\lambda x:A\,.\,\lambda y:A\,.\,x} A \to (A \to A)}$$

$$\dfrac{\dfrac{\dfrac{\overline{x:A \vdash_x A}}{x:A, y:A \vdash_x A}}{y:A \vdash_{\lambda x:A\,.\,x} A \to A}}{\vdash_{\lambda y:A\,.\,\lambda x:A\,.\,x} A \to (A \to A)}$$

□

8.2. Az izomorfizmus

Az intuícionista minimális logika természetes levezetés bizonyítási rendszere, mint az előző szakaszban láttuk, a formula bizonyítására, a formulához tartozó konstrukcióra az F_1 λ-kalkulus egy olyan kifejezését adja meg, ahol a formula a kifejezés típusa. Ez tehát azt jelenti, hogy ha az $IPC_{\to}$-ben $\Gamma \vdash_E A$, akkor és csak akkor a λ-kalkulusban $\Gamma \vdash E{:}A$.

Ennél azonban több kapcsolat is felfedezhető, a bizonyítások normalizálása, amelynek a logikában standard módszere van, éppen a λ-kifejezések redukálásának, a normalizálás szabályai a redukálás szabályainak felelnek meg.

Ezt a kapcsolatot nevezzük *Curry–Howard-izomorfizmusnak.*

8.2.1. Tétel. (Curry–Howard-izomorfimus)

Ha az F_1 elsőrendű típusos λ-kalkulusban az A típushoz van olyan zárt E λ-kifejezés, melyre $E : A$, akkor és csak akkor az A formula az $IPC_{\to}$ egy bizonyítható formulája.

A tétel tehát azt mondja ki, hogy ha $\vdash E{:}A$, akkor és csak akkor $\vdash_E A$. Mivel a tételben szereplő A formula tautológia, azaz $\emptyset \vdash_E A$, az $\emptyset \vdash E{:}A$ is fennáll, vagyis a típuskörnyezet az üres halmaz. Ebből viszont az következik, hogy az E λ-kifejezésnek zártnak kell lennie.

A Curry–Howard-izomorfizmust gyakran „formula mint típus” és „bizonyítás mint kifejezés” izomorfizmusnak is nevezik. Az első egy szintaktikus kapcsolatot ír le, a „formula mint típus” izomorfizmus azt jelenti, hogy egy A kifejezést az intuícionista minimális logika formulájának és az elsőrendű típusos λ-kalkulus egy típusának is tekinthetünk.

A „formula mint típus” izomorfizmus következő fogalmak közötti kapcsolatokat állapítja meg:

$IPC_{\to}$	λ-kalkulus
$\to$ logikai összekötőjel	$\to$ típuskonstruktor
formula	típus

A „bizonyítás mint kifejezés” izomorfizmus a természetes levezetés és a λ-kalkulus kifejezéseinek redukciós lépései közötti kapcsolatra utal.

$IPC_{\rightarrow}$	λ-kalkulus
feltétel	λ-kalkulus változója
implikáció bevezetése	absztrakció
implikáció eliminálása	applikáció
bizonyítás, konstrukció	λ-kifejezés
normalizálás	redukálás

A Curry–Howard-izomorfizmus egyik érdekes és fontos következménye az, hogy a „van-e az A formulának bizonyítása?” kérdés ekvivalens a „van-e A típusú λ-kifejezés?” kérdéssel. Finomítva a kérdést, „az A formula tautológia?” kérdés ekvivalens a „van-e A típusú zárt λ-kifejezés?” kérdéssel. A kérdésekre adott válaszok is ekvivalensek. Ha az A formulának nincs bizonyítása, akkor az A típushoz sem adható λ-kifejezés. Ha az A-nak van bizonyítása, akkor az A-hoz van λ-kifejezés, és ez a λ-kifejezés a bizonyítás lépéseiből könnyen meghatározható.

Egy hasonlóan fontos következmény az, hogy „a P bizonyítás az A formula egy bizonyítása?” kérdés ekvivalens azzal, hogy „a P típus-levezetés eredménye az, hogy egy E λ-kifejezésre $E : A$?” A bizonyítások és a típus-levezetések között már nincs ekvivalencia, mert egy A formulára több különböző bizonyítás is adható, és ezekhez a bizonyításokhoz különböző λ-kifejezések tartoznak (8.1.11. példa). Az utóbbi kérdésben tehát az E kifejezés nem egyértelmű. Ha az első kérdésre adott válasz *igen*, akkor ez csak azt jelenti, hogy van olyan E kifejezés, melyre a második kérdés is *igen* választ kap, ami éppen egy $E : A$ ellenőrzésének, azaz a típusellenőrzésnek felel meg.

Ezzel tehát az $IPC_{\rightarrow}$ és az F_1 között további két kapcsolatot kaptunk:

$IPC_{\rightarrow}$	λ-kalkulus
bizonyíthatóság	típusos kifejezés létezése
bizonyítás ellenőrzés	típusellenőrzés

A Curry–Howard-izomorfizmus jelentőségét tehát az is mutatja, hogy ami bizonyítható az F_1 rendszerben, az bizonyítható az $IPC_{\rightarrow}$-ben is, és fordítva is, azaz az egyikben bizonyított eredmény könnyen átvihető a másik rendszerbe.

A Curry–Howard-izomorfizmus kiterjeszthető az ítéletlogika további logikai összekötőjeleire is. A természetes levezetésnek a $\wedge$ és $\vee$ összekötőjelekre vonatkozó szabályai a szorzat és az összeg típust adják meg, és a *false* és *true* logikai konstansoknak a null és a unit kifejezések felelnek meg.

Az $IPC_{\rightarrow}$ és az F_1 közötti izomorfizmushoz hasonló kapcsolat található meg a λP és a predikátum (elsőrendű) logika között. Sőt mi több, Barendregt megmutatta, hogy a λ-kocka minden kalkulusához megadható egy olyan logika, amely a kalkulussal izomorf.

9. FEJEZET

Típuslevezetések

10. FEJEZET

Az $\boldsymbol{F}^{\boldsymbol{\omega}}_{\leq}$ alkalmazása

ide kell majd az OO.

11. FEJEZET

π-kalkulus

12. FEJEZET

Objektum-kalkulus

12.1. Peano axiómák

„Minden számolás alapja: a természetes számok, így nevezzük az

$0, 1, 2, 3, 4, 5, \ldots$

számokat [...] A természetes számokat és a számlálást alapfogalmaknak tekintjük, amelyeket nem definiálunk." [?] A 0 tehát természetes szám, és a számlálás műveletével minden természetes számnak meg tudjuk határozni a rákövetkezőjét.

Vajon a matematikában a természetes számok függetlenek-e, azaz tudjuk-e a természetes számokat más matematikai fogalmaktól, például a valós számoktól, vagy a végtelen halmazoktól függetlenül definiálni? Az első gondolat az, hogy a válasz csak az „igen" lehet, „kell", hogy találjunk a természetes számokra ilyen mindentől független definíciót.

Az egyik ok, hogy ilyen definíciót keresünk, a halmazelmélet paradoxonai. Ismeretes, hogy a ZF Zermelo-Fraenkel halmazelmélet [1] nem teszi lehetővé a Russel[2] -, Cantor [3] - és Burali-Forti[4]-paradoxonokat. De mi garantálja, hogy nincs, nem lehetséges több ellentmondás? A természetes számok rendszere a matematika legmegbízhatóbb ágának tünik. De mi garantálja, hogy a természetes számok aritmetikájában nem lehetnek paradoxonok?

H. Grassmann (1861), G. Frege (1884), R. Dedekind (1988) axiómatizálási

[1]...

[2]Legyen S azon halmazok halmaza, amelyek nem tartalmazzák elemként önmagukat, azaz $S \equiv \{X \mid X \notin X\}$. Ekkor $S \in S$?

[3]Legyen C az öszes halmazt tartalmazó halmaz. Ekkor C minden részhalmaza a C egy eleme, azaz $2^C \subseteq C$, és így $|2^C| \leq |C|$, ugyanakkor $|2^C| > |C|$.

[4]Ha Ω ordinálok lineérisan rendezet halmaza, akkor Ω és $\Omega + 1$ is ordinál. Ekkor, mivel $\Omega + 1 \in \Omega$, $\Omega + 1 \leq \Omega$, és mivel $\forall x \in \Omega$-ra $x + 1 > x$, $\Omega + 1 > \Omega$.

kísérletei után G. Peano (1889) adta meg a ma is használt pontos definíciókat.

12.1.1. Definíció. Peano-axiómák:

Legyenek $x, y, \ldots$ változók a természetes számok felett, legyen 0 a „nulla" szám, és jelölje succ x az $x + 1$ műveletet. Legyen továbbá F a természetes számok egy halmaza. Ekkor Peano-axiómáknak a következő három állítást nevezzük:

P1 : $\forall x\ not(0 = succ\ x)$

P2 : $\forall x \forall y\ (not(x = y) \twoheadrightarrow not(succ\ x = succ\ y))$

P3 : $(0 \in F\ and\ \forall x(x \in F \twoheadrightarrow succ\ x \in F)) \twoheadrightarrow \forall x\ (x \in F)$

A $P1$ axióma szerint a 0 egyetlen természetes számnak sem rákövetkezője, a $P2$ szerint különböző természetes számok rákövetkezői sem lehetnek egyenlőek, a $P3$ pedig a *teljes indukció* apja, ha a természetes számok egy F halmaza tartalmazza a 0-t, és ha $x \in F$-ből az következik, hogy $succ\ x \in F$, akkor az F az összes természetes számot tartammazza.

Könnyen bizonyítható, hogy két olyan N_1 és N_2 rendszer, amely kielégíti a fenti axiómákat, izomorf.

Egy ilyen rendszer megadható például a ZF halmazelméletben. Legyenek a változók egy w halmaz elemei, legyen 0 az üres halmaz, és a $succ\ x$ az $x \cup \{x\}$ művelet. Ekkor a Peano-axiómákat, mint a ZF halmazelmélet tételeit lehet bebizonyítani.

Látható, hogy a $P3$ axiómában szerepel egy halmaz-fogalom. Ezt úgy kerülhetjük el, hogy az F halmaz elemeit „példányosítjuk" például egy L formális nyelv formuláival, azaz a $P3$ axiómában módosítjuk az F halmazt egy „definiálható" halmazra.

A minimális L nyelv, ezt jelöljük L_0-val, legyen a következő: A szimbólumok legyenek az $x, y, \ldots$ változók, a 0 konstans, és a *succ* függvény, valamint az $=$ jel. Ekkor

$$\begin{array}{lll} <\mathit{kifejez\acute{e}s}> & ::= & 0 \mid x \mid y \mid \ldots \\ & \mid & succ <\mathit{kifejez\acute{e}s}> \end{array}$$

Ebben a nyelvben két típusú kifejezés van, a *succ succ* ... *succ* 0 a természetes számokat, a *succ succ* ... *succ* x típusú kifejezés az $x + n$ összeadás függvényt reprezentálja.

Az L_0 nyelv *atomi formulái* $E_1 = E_2$ alakúak, ahol E_1 és E_2 kifejezések. A nyelv *formulái* atomi formulákból épülnek fel logikai műveleteket és kvantorok felhasználásával.

Definiáljuk a Peáno-axiómákat az L_0 nyelv segítségével.

12.1.2. Definíció. A PA$_0$ Peano-axiómák:

Legyenek $x, y, \ldots$ változók a természetes számok felett, legyen 0 a „nulla" szám, és jelölje succ x az $x+1$ műveletet. Ekkor PA_0 Peano-axiómáknak a következő három állítást nevezzük:

$\mathbf{P1_0}$: $\forall x \; not(0 = succ\ x)$

$\mathbf{P2_0}$: $\forall x \forall y \; (not(x = y) \; \rightarrow \; not(succ\ x = succ\ y))$

$\mathbf{P3_0}$: $(B(0) \; and \; \forall x(B(x) \; \rightarrow \; B(succ\ x))) \; \rightarrow \; \forall x \; B(x)$,
ahol B az L_0 nyelv egy olyan formulája,
amelyben az x egy szabad változó

Látható, hogy az első két axióma változatlan maradt, csak a harmadik axióma változott meg.

A PA_0 axiómákkal definiált rendszer leíróereje nagyon kicsi, például a $x < y$ relációt sem lehet kifejezni vele.

...

FÜGGELÉK

A. FÜGGELÉK

A könyvben alkalmazott jelölések

A könyvben az ábécé betűivel a következőket jelöljük:

$x, y, z, \ldots$	változó
$E, F, G, \ldots$	kifejezés
$\alpha, \beta, \gamma, \ldots$	típusváltozó
$A, B, C, \ldots$	típuskifejezés
$\star$	típusok típusa
$K, L, \ldots$	fajta
κ	fajtaváltozó
$\Box$	fajták típusa
$\Gamma, \Delta, \ldots$	típuskörnyezet
$\mathcal{C}, \mathcal{D}$	kontextus
$X, Y, Z, \ldots$	kontextus változó

A betűváltozatok jelentése a következő:

Kurzív	típus neve (nagy kezdőbetűvel)
kurzív	matematikai függvény neve
groteszk	kifejezés neve (λ-kalkulus)
kurzív groteszk	kifejezés neve (kombinátor logika)
KISKAPITÁLIS	típusszabály neve

B. FÜGGELÉK

Típusrendszerek összehasonlító táblázatai

Rövidítések az F_ω típusrendszer leírásában:

$\Gamma \vdash A$	$\equiv$	$\Gamma \vdash A : \star$
$\Gamma \vdash K$	$\equiv$	$\Gamma \vdash K : \Box$

Rövidítések az $F^\omega_\leq$ típusrendszer leírásában:

$\Gamma \vdash A$	$\equiv$	$\Gamma \vdash A : \star$
$\Gamma \vdash K$	$\equiv$	$\Gamma \vdash K : \Box$
$\alpha : K$	$\equiv$	$\alpha \leq Top : K$
$\alpha \leq A$	$\equiv$	$\alpha \leq A : \star$
α	$\equiv$	$\alpha \leq Top : \star$
$\Gamma \vdash A : K$	$\equiv$	$\Gamma \vdash A \leq Top : K$
$\Gamma \vdash A \leq B$	$\equiv$	$\Gamma \vdash A \leq B : \star$

1. Az $\boldsymbol{F_1}$, $\boldsymbol{F_2}$ és $\boldsymbol{F_\omega}$ típusrendszer

1. Szintaxis

(a) típuskörnyezet

	F_1	F_2	F_ω
	$\Gamma ::= \emptyset$ $\mid \Gamma, x : A$	$\Gamma ::= \emptyset$ $\mid \Gamma, x : A$ $\mid \Gamma, \alpha$	$\Gamma ::= \emptyset$ $\mid \Gamma, x : A$ $\mid \Gamma, \alpha : K$

(b) fajta, típus, kifejezés

λ	F_1	F_2	F_ω
			$K ::= \star$ $\mid K \to L$
	$A ::=$ *alaptípus* $\mid A \to B$	$A ::= \alpha$ $\mid A \to B$ $\mid \forall\alpha . A$	$A ::= \alpha$ $\mid A \to B$ $\mid \forall\alpha : K . A$ $\mid \Lambda\alpha{:}K . A$ $\mid AB$
$E ::= x$ $\mid \lambda x . E$ $\mid EF$	$E ::= x$ $\mid \lambda x : A . E$ $\mid EF$	$E ::= x$ $\mid \lambda x{:}A . E$ $\mid EF$ $\mid \Lambda\alpha . E$ $\mid E\,[\,A\,]$	$E ::= x$ $\mid \lambda x{:}A . E$ $\mid EF$ $\mid \Lambda\alpha : K . E$ $\mid E\,[\,A\,]$

2. Redukciók			
$\boldsymbol{\lambda}$	$\boldsymbol{F_1}$	$\boldsymbol{F_2}$	$\boldsymbol{F_\omega}$
$(\lambda x\,.\,E)F \twoheadrightarrow_\beta E\,[\,x := F\,]$	$(\lambda x\,{:}A\,.\,E)F \twoheadrightarrow_\beta E\,[\,x := F\,]$	$(\lambda x{:}A\,.\,E)F \twoheadrightarrow_\beta E\,[\,x := F\,]$ $(\Lambda\alpha\,.\,E)\,[\,A\,] \twoheadrightarrow_\beta E\,[\,\alpha := A\,]$	$(\lambda x{:}A\,.\,E)F \twoheadrightarrow_\beta E\,[\,x := F\,]$ $(\Lambda\alpha\,{:}K\,.\,E)\,[\,A\,] \twoheadrightarrow_\beta E\,[\,\alpha := A\,]$
3. Következtetések			
	$\boldsymbol{F_1}$	$\boldsymbol{F_2}$	$\boldsymbol{F_\omega}$
	$\Gamma \vdash wf$ $\vdash A$ $\vdash E : A$	$\Gamma \vdash wf$ $\vdash A$ $\vdash E : A$	$\Gamma \vdash wf$ $\vdash A : K$ $\vdash E : A$ $\vdash K$

4. Szabályok

(a) típuskörnyezet

F_1	F_2	F_ω	
$\dfrac{}{\emptyset \vdash wf}$	$\dfrac{}{\emptyset \vdash wf}$	$\dfrac{}{\emptyset \vdash wf}$	Env $\emptyset$
$\dfrac{\Gamma \vdash A \quad x \notin dom(\Gamma)}{\Gamma, x{:}A \vdash wf}$	$\dfrac{\Gamma \vdash A \quad x \notin dom(\Gamma)}{\Gamma, x{:}A \vdash wf}$	$\dfrac{\Gamma \vdash A \quad x \notin dom(\Gamma)}{\Gamma, x{:}A \vdash wf}$	Env x
	$\dfrac{\Gamma \vdash wf \quad \alpha \notin dom(\Gamma)}{\Gamma, \alpha \vdash wf}$	$\dfrac{\Gamma \vdash K \quad \alpha \notin dom(\Gamma)}{\Gamma, \alpha : K \vdash wf}$	Env α

(b) fajta

F_1	F_2	F_ω	
		$\dfrac{\Gamma \vdash wf}{\Gamma \vdash \star}$	Kind $\star$
		$\dfrac{\Gamma \vdash K \quad \Gamma \vdash L}{\Gamma \vdash K \to L}$	Kind Arrow
		$\dfrac{\Gamma, \alpha : K \vdash A : L}{\Gamma \vdash \Lambda\alpha{:}K\,.\,A : K \to L}$	Kind Fun
		$\dfrac{\Gamma \vdash A : K \to L \quad \Gamma \vdash B : K}{\Gamma \vdash AB : L}$	Kind Appl

(c) típus

F_1	F_2	F_ω	
$\dfrac{\Gamma \vdash \mathit{wf} \quad A \in \mathit{alaptípus}}{\Gamma \vdash A}$			TYPE CONST
	$\dfrac{\Gamma', \alpha, \Gamma'' \vdash \mathit{wf}}{\Gamma', \alpha, \Gamma'' \vdash \alpha}$	$\dfrac{\Gamma', \alpha : K, \Gamma'' \vdash \mathit{wf}}{\Gamma', \alpha : K, \Gamma'' \vdash \alpha : K}$	TYPE α
$\dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \to B}$	$\dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \to B}$	$\dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \to B}$	TYPE ARROW
	$\dfrac{\Gamma, \alpha \vdash A}{\Gamma \vdash \forall\alpha \,.\, A}$	$\dfrac{\Gamma, \alpha : K \vdash A}{\Gamma \vdash \forall\alpha{:}K \,.\, A}$	TYPE FORALL

(d) kifejezés

F_1	F_2	F_ω	
$\dfrac{\Gamma', x : A, \Gamma'' \vdash \mathit{wf}}{\Gamma', x : A, \Gamma'' \vdash x : A}$	$\dfrac{\Gamma', x : A, \Gamma'' \vdash \mathit{wf}}{\Gamma', x : A, \Gamma'' \vdash x : A}$	$\dfrac{\Gamma', x : A, \Gamma'' \vdash \mathit{wf}}{\Gamma', x : A, \Gamma'' \vdash x : A}$	VAL x
$\dfrac{\Gamma, x : A \vdash E : B}{\Gamma \vdash \lambda x{:}A \,.\, E : A \to B}$	$\dfrac{\Gamma, x : A \vdash E : B}{\Gamma \vdash \lambda x{:}A \,.\, E : A \to B}$	$\dfrac{\Gamma, x : A \vdash E : B}{\Gamma \vdash \lambda x{:}A \,.\, E : A \to B}$	VAL FUN
$\dfrac{\Gamma \vdash E : A \to B \quad \Gamma \vdash F : A}{\Gamma \vdash EF : B}$	$\dfrac{\Gamma \vdash E : A \to B \quad \Gamma \vdash F : A}{\Gamma \vdash EF : B}$	$\dfrac{\Gamma \vdash E : A \to B \quad \Gamma \vdash F : A}{\Gamma \vdash EF : B}$	VAL APPL
	$\dfrac{\Gamma, \alpha \vdash E : A}{\Gamma \vdash \Lambda\alpha \,.\, E : \forall\alpha \,.\, A}$	$\dfrac{\Gamma, \alpha : K \vdash E : A}{\Gamma \vdash \Lambda\alpha{:}K \,.\, E : \forall\alpha{:}K \,.\, A}$	VAL FUN2
	$\dfrac{\Gamma \vdash E : \forall\alpha \,.\, A \quad \Gamma \vdash B}{\Gamma \vdash E[B] : A[\alpha := B]}$	$\dfrac{\Gamma \vdash E : \forall\alpha{:}K \,.\, A \quad \Gamma \vdash B : K}{\Gamma \vdash E[B] : A[\alpha := B]}$	VAL APPL2

2. Az F_1 és λP típusrendszer

1. Szintaxis	
(a) típuskörnyezet	
F_1	λP
$\boxed{\Gamma} ::= \boxed{\emptyset} \mid \boxed{\Gamma, x : A}$	$\Gamma ::= \emptyset \mid \Gamma, x : A$
(b) típus, kifejezés	
F_1	λP
$\boxed{A} ::= \boxed{\textit{alaptípus}} \mid \boxed{A \to B}$	$A ::= \textit{alaptípus} \mid A \to B \mid \boxed{\Pi x : A \,.\, B}$
$\boxed{E} ::= \boxed{x} \mid \boxed{\lambda x : A \,.\, E} \mid \boxed{EF}$	$E ::= x \mid \lambda x : A \,.\, E \mid EF$

2. Redukciók

F_1	λP
$(\lambda x\,.\,E)F \;\rightarrow_\beta$ $E\,[\,x := F\,]$	$(\lambda x\,.\,E)F \;\rightarrow_\beta$ $E\,[\,x := F\,]$

3. Következtetések

F_1	λP
$\Gamma \;\vdash\; wf$ $\vdash\; A$ $\vdash\; E\!:\!A$	$\Gamma \;\vdash\; wf$ $\vdash\; A$ $\vdash\; E\!:\!A$

4. Szabályok

(a) típuskörnyezet

F_1	λP	
$\dfrac{}{\emptyset \vdash wf}$	$\dfrac{}{\emptyset \vdash wf}$	Env $\emptyset$
$\dfrac{\Gamma \vdash A \quad x \notin dom(\Gamma)}{\Gamma, x\!:\!A \vdash wf}$	$\dfrac{\Gamma \vdash A \quad x \notin dom(\Gamma)}{\Gamma, x\!:\!A \vdash wf}$	Env x

(c) típus

$\boldsymbol{F_1}$	$\boldsymbol{\lambda P}$	
$\dfrac{\Gamma \vdash \mathit{wf} \quad A \in \mathit{alaptípus}}{\Gamma \vdash A}$	$\dfrac{\Gamma \vdash \mathit{wf} \quad A \in \mathit{alaptípus}}{\Gamma \vdash A}$	TYPE CONST
$\dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \to B}$	$\dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \to B}$	TYPE ARROW
	$\dfrac{\Gamma, x : A \vdash B}{\Gamma \vdash \Pi x{:}A\,.\,B}$	TYPE DEP

(d) kifejezés

$\boldsymbol{F_1}$	$\boldsymbol{\lambda P}$	
$\dfrac{\Gamma', x : A, \Gamma'' \vdash \mathit{wf}}{\Gamma', x : A, \Gamma'' \vdash x : A}$	$\dfrac{\Gamma', x : A, \Gamma'' \vdash \mathit{wf}}{\Gamma', x : A, \Gamma'' \vdash x : A}$	VAL x
$\dfrac{\Gamma, x : A \vdash E : B}{\Gamma \vdash \lambda x{:}A\,.\,E : A \to B}$	$\dfrac{\Gamma, x : A \vdash E : B}{\Gamma \vdash \lambda x{:}A\,.\,E : A \to B}$	VAL FUN
$\dfrac{\Gamma \vdash E : A \to B \quad \Gamma \vdash F : A}{\Gamma \vdash EF : B}$	$\dfrac{\Gamma \vdash E : A \to B \quad \Gamma \vdash F : A}{\Gamma \vdash EF : B}$	VAL APPL
	$\dfrac{\Gamma, x : A \vdash E : B}{\Gamma \vdash \lambda x{:}A\,.\,E : \Pi x{:}A\,.\,B}$	VAL DEP
	$\dfrac{\Gamma \vdash E : \Pi x{:}A\,.\,B \quad \Gamma \vdash F : A}{\Gamma \vdash EF : B\,[\,x := F\,]}$	VAL DEP APPL

3. Az F_2, $F_{2\exists}$ és $F_{2\mu}$ típusrendszer

1. Szintaxis		
(a) típuskörnyezet		
F_2	$F_{2\exists}$	$F_{2\mu}$
$\Gamma ::= \emptyset$	$\Gamma ::= \emptyset$	$\Gamma ::= \emptyset$
$\mid \Gamma, x : A$	$\mid \Gamma, x : A$	$\mid \Gamma, x : A$
$\mid \Gamma, \alpha$	$\mid \Gamma, \alpha$	$\mid \Gamma, \alpha$
(b) típus, kifejezés		
F_2	$F_{2\exists}$	$F_{2\mu}$
$A ::= \alpha$	$A ::= \alpha$	$A ::= \alpha$
$\mid A \to B$	$\mid A \to B$	$\mid A \to B$
$\mid \forall\alpha \,.\, A$	$\mid \forall\alpha \,.\, A$	$\mid \forall\alpha \,.\, A$
	$\mid \exists\alpha \,.\, A$	
		$\mid \mu\alpha \,.\, A$
$E ::= x$	$E ::= x$	$E ::= x$
$\mid \lambda x{:}A \,.\, E$	$\mid \lambda x{:}A \,.\, E$	$\mid \lambda x{:}A \,.\, E$
$\mid E F$	$\mid E F$	$\mid E F$
$\mid \Lambda\alpha \,.\, E$	$\mid \Lambda\alpha \,.\, E$	$\mid \Lambda\alpha \,.\, E$
$\mid E\,[\,A\,]$	$\mid E\,[\,A\,]$	$\mid E\,[\,A\,]$
	$\mid$ pack A with E	
	$\mid$ unpack α, x from E in F	
		$\mid$ fold $[\,A\,]\,E$
		$\mid$ unfold $[\,A\,]\,E$

2. Redukciók		
$\boldsymbol{F_2}$	$\boldsymbol{F_{2\exists}}$	$\boldsymbol{F_{2\mu}}$
$(\lambda x{:}A\,.\,E)F \twoheadrightarrow_\beta E\,[\,x := F\,]$	$(\lambda x{:}A\,.\,E)F \twoheadrightarrow_\beta E\,[\,x := F\,]$	$(\lambda x{:}A\,.\,E)F \twoheadrightarrow_\beta E\,[\,x := F\,]$
$(\Lambda\alpha\,.\,E)\,[\,A\,] \twoheadrightarrow_\beta E\,[\,\alpha := A\,]$	$(\Lambda\alpha\,.\,E)\,[\,A\,] \twoheadrightarrow_\beta E\,[\,\alpha := A\,]$	$(\Lambda\alpha\,.\,E)\,[\,A\,] \twoheadrightarrow_\beta E\,[\,\alpha := A\,]$

3. Következtetések		
$\boldsymbol{F_2}$	$\boldsymbol{F_{2\exists}}$	$\boldsymbol{F_{2\mu}}$
$\Gamma \vdash wf$	$\Gamma \vdash wf$	$\Gamma \vdash wf$
$\vdash A$	$\vdash A$	$\vdash A$
$\vdash E : A$	$\vdash E : A$	$\vdash E : A$

4. Szabályok			
(a) típuskörnyezet			
$\boldsymbol{F_2}$	$\boldsymbol{F_{2\exists}}$	$\boldsymbol{F_{2\mu}}$	
$\dfrac{}{\emptyset \vdash wf}$	$\dfrac{}{\emptyset \vdash wf}$	$\dfrac{}{\emptyset \vdash wf}$	Env $\emptyset$
$\dfrac{\Gamma \vdash A \quad x \notin dom(\Gamma)}{\Gamma, x : A \vdash wf}$	$\dfrac{\Gamma \vdash A \quad x \notin dom(\Gamma)}{\Gamma, x : A \vdash wf}$	$\dfrac{\Gamma \vdash A \quad x \notin dom(\Gamma)}{\Gamma, x : A \vdash wf}$	Env x
$\dfrac{\Gamma \vdash A \quad \alpha \notin dom(\Gamma)}{\Gamma, \alpha : A \vdash wf}$	$\dfrac{\Gamma \vdash A \quad \alpha \notin dom(\Gamma)}{\Gamma, \alpha : A \vdash wf}$	$\dfrac{\Gamma \vdash A \quad \alpha \notin dom(\Gamma)}{\Gamma, \alpha : A \vdash wf}$	Env α

(b) típus			
$\boldsymbol{F_2}$	$\boldsymbol{F_{2\exists}}$	$\boldsymbol{F_{2\mu}}$	
$\dfrac{\Gamma', \alpha, \Gamma'' \vdash \ wf}{\Gamma', \alpha, \Gamma'' \vdash \alpha}$	$\dfrac{\Gamma', \alpha, \Gamma'' \vdash \ wf}{\Gamma', \alpha, \Gamma'' \vdash \alpha}$	$\dfrac{\Gamma', \alpha, \Gamma'' \vdash \ wf}{\Gamma', \alpha, \Gamma'' \vdash \alpha}$	TYPE α
$\dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \to B}$	$\dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \to B}$	$\dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \to B}$	TYPE ARROW
$\dfrac{\Gamma, \alpha \vdash A}{\Gamma \vdash \forall\alpha\,.\,A}$	$\dfrac{\Gamma, \alpha \vdash A}{\Gamma \vdash \forall\alpha\,.\,A}$	$\dfrac{\Gamma, \alpha \vdash A}{\Gamma \vdash \forall\alpha\,.\,A}$	TYPE FORALL
	$\dfrac{\Gamma, \alpha \vdash A}{\Gamma \vdash \exists\alpha\,.\,A}$		TYPE EXISTS
		$\dfrac{\Gamma, \alpha \vdash A}{\Gamma \vdash \mu\alpha\,.\,A}$	TYPE REC

(c) kifejezés

F_2	$F_{2\exists}$	$F_{2\mu}$	
$\dfrac{\Gamma', x : A, \Gamma'' \vdash wf}{\Gamma', x : A, \Gamma'' \vdash x : A}$	$\dfrac{\Gamma', x : A, \Gamma'' \vdash wf}{\Gamma', x : A, \Gamma'' \vdash x : A}$	$\dfrac{\Gamma', x : A, \Gamma'' \vdash wf}{\Gamma', x : A, \Gamma'' \vdash x : A}$	VAL x
$\dfrac{\Gamma, x : A \vdash E : B}{\Gamma \vdash \lambda x{:}A\,.\,E : A \to B}$	$\dfrac{\Gamma, x : A \vdash E : B}{\Gamma \vdash \lambda x{:}A\,.\,E : A \to B}$	$\dfrac{\Gamma, x : A \vdash E : B}{\Gamma \vdash \lambda x{:}A\,.\,E : A \to B}$	VAL FUN
$\dfrac{\Gamma \vdash E : A \to B \quad \Gamma \vdash F : A}{\Gamma \vdash EF : B}$	$\dfrac{\Gamma \vdash E : A \to B \quad \Gamma \vdash F : A}{\Gamma \vdash EF : B}$	$\dfrac{\Gamma \vdash E : A \to B \quad \Gamma \vdash F : A}{\Gamma \vdash EF : B}$	VAL APPL
$\dfrac{\Gamma, \alpha \vdash E : A}{\Gamma \vdash \Lambda\alpha\,.\,E : \forall\alpha\,.\,A}$	$\dfrac{\Gamma, \alpha \vdash E : A}{\Gamma \vdash \Lambda\alpha\,.\,E : \forall\alpha\,.\,A}$	$\dfrac{\Gamma, \alpha \vdash E : A}{\Gamma \vdash \Lambda\alpha\,.\,E : \forall\alpha\,.\,A}$	VAL FUN2
$\dfrac{\Gamma \vdash E : \forall\alpha\,.\,A \quad \Gamma \vdash B}{\Gamma \vdash E[B] : A[\alpha := B]}$	$\dfrac{\Gamma \vdash E : \forall\alpha\,.\,A \quad \Gamma \vdash B : K}{\Gamma \vdash E[B] : A[\alpha := B]}$	$\dfrac{\Gamma \vdash E : \forall\alpha\,.\,A \quad \Gamma \vdash B : K}{\Gamma \vdash E[B] : A[\alpha := B]}$	VAL APPL2
	$\dfrac{\Gamma \vdash B \quad \Gamma \vdash E : A[\alpha := B]}{\Gamma \vdash \textsf{pack}\ B\ \textsf{with}\ E : \exists\alpha\,.\,A}$		VAL PACK
	$\dfrac{\Gamma \vdash \beta \quad \Gamma \vdash F : \exists\alpha\,.\,A \quad \Gamma, x : A[\alpha := \beta] \vdash G : C \quad \beta \notin FV(C)}{\Gamma \vdash (\textsf{unpack}\ \beta, x\ \textsf{from}\ F\ \textsf{in}\ G) : C}$		VAL UNPACK
		$\dfrac{\Gamma \vdash E{:}A[\alpha := \mu\alpha\,.\,A]}{\Gamma \vdash \textsf{fold}\,[\mu\alpha\,.\,A]\,E : \mu\alpha\,.\,A}$	VAL FOLD
		$\dfrac{\Gamma \vdash E{:}\mu\alpha\,.\,A}{\Gamma \vdash \textsf{unfold}\,[\mu\alpha\,.\,A]\,E : A[\alpha := \mu\alpha\,.\,A]}$	VAL UNFOLD

4. Az $\boldsymbol{F_\omega}$ és az $\boldsymbol{F^\omega_\leq}$ típusrendszer

1. Szintaxis

(a) típuskörnyezet

$\boldsymbol{F_\omega}$	$\boldsymbol{F^\omega_\leq}$
$\Gamma \ ::= \ \emptyset$	$\Gamma \ ::= \ \emptyset$
$\mid \ \Gamma, x : A$	$\mid \ \Gamma, x : A$
$\mid \ \Gamma, \alpha$	$\mid \ \Gamma, \alpha \leq K$

(b) fajta, típus, kifejezés

$\boldsymbol{F_\omega}$	$\boldsymbol{F^\omega_\leq}$
$K \ ::= \ \star$	$K \ ::= \ \star$
$\mid \ K \to L$	$\mid \ K \to L$
$A \ ::= \ \alpha$	$A \ ::= \ \alpha$
	$\mid \ \mathit{Top}$
$\mid \ A \to B$	$\mid \ A \to B$
$\mid \ \forall \alpha{:}K\,.\,A$	$\mid \ \forall \alpha \leq A{:}K\,.\,B$
$\mid \ \Lambda \alpha{:}K\,.\,A$	$\mid \ \Lambda \alpha \leq A{:}K\,.\,B$
$\mid \ AB$	$\mid \ AB$
$E \ ::= \ x$	$E \ ::= \ x$
$\mid \ \lambda x{:}A\,.\,E$	$\mid \ \lambda x{:}A\,.\,E$
$\mid \ EF$	$\mid \ EF$
$\mid \ \Lambda \alpha{:}K\,.\,E$	$\mid \ \Lambda \alpha \leq A{:}K\,.\,E$
$\mid \ E\,[A]$	$\mid \ E\,[A]$

2. Redukciók

F_ω	$F^\omega_\leq$
$(\lambda x \,.\, E)F \;\twoheadrightarrow_\beta E\,[\,x := F\,]$	$(\lambda x \; {:}A \,.\, E)F \;\twoheadrightarrow_\beta E\,[\,x := F\,]$
$(\Lambda\alpha \,.\, E)\,[\,A\,] \;\twoheadrightarrow_\beta E\,[\,\alpha := A\,]$	$(\Lambda\alpha \; {:}K \,.\, E)\,[\,A\,] \;\twoheadrightarrow_\beta E\,[\,\alpha := A\,]$

3. Következtetések

F_ω	$F^\omega_\leq$
$\Gamma \vdash wf$	$\Gamma \vdash wf$
$\vdash K$	$\vdash K$
$\vdash A : K$	$\vdash A : K$
	$\vdash A \leq B : K$
$\vdash E : A$	$\vdash E : A$

4. Szabályok

(a) típuskörnyezet

F_ω	$F^\omega_{\leq}$	
$\dfrac{}{\emptyset \vdash wf}$	$\dfrac{}{\emptyset \vdash wf}$	Env $\emptyset$
$\dfrac{\Gamma \vdash A \quad x \notin dom(\Gamma)}{\Gamma, x : A \vdash wf}$	$\dfrac{\Gamma \vdash A \quad x \notin dom(\Gamma)}{\Gamma, x : A \vdash wf}$	Env x
$\dfrac{\Gamma \vdash A : K \quad \alpha \notin dom(\Gamma)}{\Gamma, \alpha : A \vdash wf}$	$\dfrac{\Gamma \vdash A : K \quad \alpha \notin dom(\Gamma)}{\Gamma, \alpha \leq A : K \vdash wf}$	Env α

(b) fajta

F_ω	$F^\omega_{\leq}$	
$\dfrac{\Gamma \vdash wf}{\Gamma \vdash \star}$	$\dfrac{\Gamma \vdash wf}{\Gamma \vdash \star : \Box}$	Kind $\star$
$\dfrac{\Gamma \vdash K \quad \Gamma \vdash L}{\Gamma \vdash K \to L : \Box}$	$\dfrac{\Gamma \vdash K \quad \Gamma \vdash L}{\Gamma \vdash K \to L : \Box}$	Kind Arrow
$\dfrac{\Gamma, \alpha : K \vdash B : L}{\Gamma \vdash \Lambda\alpha{:}K\,.\,B : K \to L}$	$\dfrac{\Gamma, \alpha \leq A : K \vdash B : L}{\Gamma \vdash \Lambda\alpha \leq A{:}K\,.\,B : K \to L}$	Kind Fun
$\dfrac{\Gamma \vdash A : K \to L \quad \Gamma \vdash B : K}{\Gamma \vdash AB : L}$	$\dfrac{\Gamma \vdash A : K \to L \quad \Gamma \vdash B : K}{\Gamma \vdash AB : L}$	Kind Appl

(c) típus

F_ω	$F^\omega_\leq$	
$\dfrac{\Gamma', \alpha : K, \Gamma'' \vdash wf}{\Gamma', \alpha : K, \Gamma'' \vdash \alpha : K}$	$\dfrac{\Gamma', \alpha \leq A : K, \Gamma'' \vdash wf}{\Gamma', \alpha \leq A : K, \Gamma'' \vdash \alpha : K}$	Type α
	$\dfrac{\Gamma \vdash wf}{\Gamma \vdash Top : K}$	Type Top
$\dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \to B}$	$\dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \to B}$	Type Arrow
$\dfrac{\Gamma, \alpha : K \vdash B}{\Gamma \vdash \forall \alpha{:}K . B}$	$\dfrac{\Gamma, \alpha \leq A : K \vdash B}{\Gamma \vdash \forall \alpha \leq A : K . B}$	Type Forall

(d) altípus

F_ω	$F^\omega_\leq$	
	$\dfrac{\Gamma \vdash A : K}{\Gamma \vdash A \leq A : K}$	Sub Refl
	$\dfrac{\Gamma \vdash A \leq B : K \quad \Gamma \vdash B \leq C : K}{\Gamma \vdash A \leq C : K}$	Sub Trans
	$\dfrac{\Gamma', \alpha{\leq}A : K, \Gamma'' \vdash wf}{\Gamma', \alpha{\leq}A : K, \Gamma'' \vdash \alpha{\leq}A : K}$	Sub α
	$\dfrac{\Gamma \vdash A : K}{\Gamma \vdash A \leq Top : K}$	Sub Top
	$\dfrac{\Gamma \vdash A' \leq A \quad \Gamma \vdash B \leq B'}{\Gamma \vdash A \to B \leq A' \to B'}$	Sub Arrow
	$\dfrac{\Gamma, \alpha{\leq}A : K \vdash B \leq C : L}{\Gamma \vdash (\Lambda\alpha{\leq}A{:}K . B) \leq (\Lambda\alpha{\leq}A{:}K . C) : K \to L}$	Sub Arrow2
	$\dfrac{\Gamma \vdash A \leq A' : K \to L \quad \Gamma \vdash B : K}{\Gamma \vdash AB \leq A'B : L}$	Sub Appl2
	$\dfrac{\Gamma \vdash A' \leq A : K \quad \Gamma \vdash \alpha{\leq}A' : K \vdash B \leq B'}{\Gamma \vdash (\forall\alpha{\leq}A{:}K . B) \leq (\forall\alpha{\leq}A'{:}K . B')}$	Sub Forall

(e) kifejezés		
$\boldsymbol{F_\omega}$	$\boldsymbol{F^\omega_\leq}$	
$\dfrac{\Gamma', x : A, \Gamma'' \vdash wf}{\Gamma', x : A, \Gamma'' \vdash x : A}$	$\dfrac{\Gamma', x : A, \Gamma'' \vdash wf}{\Gamma', x : A, \Gamma'' \vdash x : A}$	Val x
$\dfrac{\Gamma, x : A \vdash E : B}{\Gamma \vdash \lambda x{:}A\,.\,E : A \to B}$	$\dfrac{\Gamma, x : A \vdash E : B}{\Gamma \vdash \lambda x{:}A\,.\,E : A \to B}$	Val Fun
$\dfrac{\Gamma \vdash E : A \to B \quad \Gamma \vdash F : A}{\Gamma \vdash EF : B}$	$\dfrac{\Gamma \vdash E : A \to B \quad \Gamma \vdash F : A}{\Gamma \vdash EF : B}$	Val Appl
$\dfrac{\Gamma, \alpha : K \vdash E : B}{\Gamma \vdash \Lambda\alpha{:}K\,.\,E : \forall\alpha{:}K\,.\,B}$	$\dfrac{\Gamma, \alpha \leq A : K \vdash E : B}{\Gamma \vdash \Lambda\alpha \leq A : K\,.\,E : \forall\alpha \leq A : K\,.\,B}$	Val Fun2
$\dfrac{\Gamma \vdash E : \forall\alpha{:}K\,.\,B \quad \Gamma \vdash C : K}{\Gamma \vdash E[C] : B[\alpha := C]}$	$\dfrac{\Gamma \vdash E : \forall\alpha \leq A : K\,.\,B \quad \Gamma \vdash C \leq A : K}{\Gamma \vdash E[C] : B[\alpha := C]}$	Val Appl2
	$\dfrac{\Gamma \vdash E : A \quad \Gamma \vdash A \leq B}{\Gamma \vdash E : B}$	Val Subsumption

B.0.1. A fajták típusa

Az F_ω kalkulusban eddig a fajta $\star$, vagy $K \to L$ alakú lehetett. A $K \to L$ fajta – mint az F_1 kalkulusban az $A \to B$ típus, – egy leképezést valósít meg, azaz egy olyan függvénynek felel meg, amelyik a K fajtát az L fajtára képezi le.

Erre a függvényre egy függvényabsztrakciót is megadhatunk. Jelöljük ezt az absztrakciót Π-vel, a függvény változóját, azaz a ***fajtaváltozót*** κ-val, így a függvény alakja

$\Pi\kappa{:}K\,.\,L.$

Ekkor mondhatjuk, hogy a κ fajtaváltozó ***hatásköre*** az L, és beszélhetünk az L-beli ***szabad*** és ***kötött*** fajtaváltozókról is.

Ha az L nem függ a κ-tól, azaz az L-ben nincs szabad κ fajtaváltozó, akkor

$\Pi\kappa{:}K\,.\,L \equiv K \to L.$

Ha a fajták típusát $\Box$-val jelöljük, akkor nyilvánvalóan

$\star : \Box,$
$(\Pi\kappa{:}K\,.\,L) : \Box.$

Ezeket a tulajdonságokat felhasználva, az F_ω és az $F^\omega_\leq$ kalkulusok szabályainak leírásaiban a $K \to L$ fajta minden esetben a $\Pi\kappa{:}K\,.\,L$, illetve a $\Pi\kappa{\leq}A{:}K\,.\,L$ kifejezéssel helyettesíthető.

F_ω	$F^\omega_\leq$	
$\dfrac{\Gamma \vdash wf}{\Gamma \vdash \star}$	$\dfrac{\Gamma \vdash wf}{\Gamma \vdash \star : \Box}$	KIND $\star$
$\dfrac{\Gamma, \kappa : K \vdash L : \Box}{\Gamma \vdash \Pi\kappa{:}K\,.\,L : \Box}$	$\dfrac{\Gamma, \kappa \leq A : K \vdash L : \Box}{\Gamma \vdash \Pi\kappa \leq A : K\,.\,L : \Box}$	KIND ARROW
$\dfrac{\Gamma, \kappa : K \vdash B : L}{\Gamma \vdash \Lambda\kappa{:}K\,.\,B : \Pi\kappa{:}K\,.\,L}$	$\dfrac{\Gamma, \kappa \leq A : K \vdash B : L}{\Gamma \vdash \Lambda\kappa \leq A : K\,.\,B : \Pi\kappa \leq A : K\,.\,L}$	KIND FUN
$\dfrac{\Gamma \vdash A : \Pi\kappa{:}K\,.\,L \quad \Gamma \vdash B : K}{\Gamma \vdash AB : L}$	$\dfrac{\Gamma \vdash A : \Pi\kappa \leq A : K\,.\,L \quad \Gamma \vdash B : K}{\Gamma \vdash AB : L}$	KIND APPL
	$\dfrac{\Gamma, \kappa{\leq}A{:}K \vdash B \leq C : L}{\Gamma \vdash (\Lambda\kappa{\leq}A{:}K\,.\,B) \leq (\Lambda\kappa{\leq}A{:}K\,.\,C) : \Pi\kappa{\leq}A{:}K\,.\,L}$	SUB ARROW2

C. FÜGGELÉK

A típusrendszerek összefoglaló leírása

I. A TÍPUSRENDSZEREK SZINTAXISA

1. A típusnélküli $\boldsymbol{\lambda}$-kalkulus szintaxisa:

$$\begin{array}{lcl}
< \lambda\text{-}\textit{kifejezés} > & ::= & < \textit{változó} > \\
 & | & \lambda < \textit{változó} > . < \lambda\text{-}\textit{kifejezés} > \\
 & | & < \lambda\text{-}\textit{kifejezés} > < \lambda\text{-}\textit{kifejezés} >
\end{array}$$

2. Az $\boldsymbol{F_1}$ rendszer szintaxisa:

$$\begin{array}{lcl}
\boldsymbol{<}\textbf{\textit{típus}}\boldsymbol{>} & ::= & \boldsymbol{<}\textbf{\textit{alaptípus}}\boldsymbol{>} \\
 & | & \boldsymbol{<}\textbf{\textit{típus}}\boldsymbol{>} \boldsymbol{\rightarrow} \boldsymbol{<}\textbf{\textit{típus}}\boldsymbol{>}
\end{array}$$

$$\begin{array}{lcl}
< \lambda\text{-}\textit{kifejezés} > & ::= & < \textit{változó} > \\
 & | & \lambda < \textit{változó} > \boldsymbol{:} \boldsymbol{<}\textbf{\textit{típus}}\boldsymbol{>} . < \lambda\text{-}\textit{kifejezés} > \\
 & | & < \lambda\text{-}\textit{kifejezés} > < \lambda\text{-}\textit{kifejezés} >
\end{array}$$

3. Az F_2 rendszer szintaxisa:

$$
\begin{array}{lll}
<\textit{típus}> & ::= & \boldsymbol{<}\textbf{\textit{típusváltozó}}\boldsymbol{>} \\
 & | & <\textit{típus}> \to <\textit{típus}> \\
 & | & \forall\, \boldsymbol{<}\textbf{\textit{típusváltozó}}\boldsymbol{>}\ \textbf{.}\ \boldsymbol{<}\textbf{\textit{típus}}\boldsymbol{>} \\
 & & \\
<\lambda\textit{-kifejezés}> & ::= & <\textit{változó}> \\
 & | & \lambda\ <\textit{változó}> : <\textit{típus}>\ .\ <\lambda\textit{-kifejezés}> \\
 & | & <\lambda\textit{-kifejezés}>\ <\lambda\textit{-kifejezés}> \\
 & | & \boldsymbol{\Lambda}\, \boldsymbol{<}\textbf{\textit{típusváltozó}}\boldsymbol{>}\ \textbf{.}\ \boldsymbol{<\lambda}\textbf{\textit{-kifejezés}}\boldsymbol{>} \\
 & | & \boldsymbol{<\lambda}\textbf{\textit{-kifejezés}}\boldsymbol{>}\ [\ \boldsymbol{<}\textbf{\textit{típus}}\boldsymbol{>}\ ]
\end{array}
$$

4. Az F_3 rendszer szintaxisa:

$$
\begin{array}{lll}
\boldsymbol{<}\textbf{\textit{fajta}}\boldsymbol{>} & ::= & * \\
 & | & \boldsymbol{*} \boldsymbol{\to} \boldsymbol{<}\textbf{\textit{fajta}}\boldsymbol{>} \\
 & & \\
<\textit{típus}> & ::= & <\textit{típusváltozó}> \\
 & | & <\textit{típus}> \to <\textit{típus}> \\
 & | & \forall\ <\textit{típusváltozó}>\ \textbf{:}\ \boldsymbol{<}\textbf{\textit{fajta}}\boldsymbol{>}\ .\ <\textit{típus}> \\
 & | & \boldsymbol{\Lambda}\, \boldsymbol{<}\textbf{\textit{típusváltozó}}\boldsymbol{>}\ \textbf{.}\ \boldsymbol{<}\textbf{\textit{típus}}\boldsymbol{>} \\
 & | & <\textit{típus}>\ <\textit{típus}> \\
 & & \\
<\lambda\textit{-kifejezés}> & ::= & <\textit{változó}> \\
 & | & \lambda\ <\textit{változó}> : <\textit{típus}>\ .\ <\lambda\textit{-kifejezés}> \\
 & | & <\lambda\textit{-kifejezés}>\ <\lambda\textit{-kifejezés}> \\
 & | & \Lambda\ <\textit{típusváltozó}>\ \textbf{:}\ \boldsymbol{<}\textbf{\textit{fajta}}\boldsymbol{>}\ .\ <\lambda\textit{-kifejezés}> \\
 & | & <\lambda\textit{-kifejezés}>\ [\ <\textit{típus}>\ ]
\end{array}
$$

5. Az F_ω rendszer szintaxisa:

$$
\begin{array}{lcl}
\langle \textit{fajta} \rangle & ::= & \star \\
 & | & \langle \textit{fajta} \rangle \to \langle \textit{fajta} \rangle \\
 & & \\
\langle \textit{típus} \rangle & ::= & \langle \textit{típusváltozó} \rangle \\
 & | & \langle \textit{típus} \rangle \to \langle \textit{típus} \rangle \\
 & | & \forall \, \langle \textit{típusváltozó} \rangle : \langle \textit{fajta} \rangle \, . \, \langle \textit{típus} \rangle \\
 & | & \Lambda \, \langle \textit{típusváltozó} \rangle \, . \, \langle \textit{típus} \rangle \\
 & | & \langle \textit{típus} \rangle \, \langle \textit{típus} \rangle \\
 & & \\
\langle \lambda\textit{-kifejezés} \rangle & ::= & \langle \textit{változó} \rangle \\
 & | & \lambda \, \langle \textit{változó} \rangle : \langle \textit{típus} \rangle \, . \, \langle \lambda\textit{-kifejezés} \rangle \\
 & | & \langle \lambda\textit{-kifejezés} \rangle \, \langle \lambda\textit{-kifejezés} \rangle \\
 & | & \Lambda \, \langle \textit{típusváltozó} \rangle : \langle \textit{fajta} \rangle \, . \, \langle \lambda\textit{-kifejezés} \rangle \\
 & | & \langle \lambda\textit{-kifejezés} \rangle \, [\, \langle \textit{típus} \rangle \,]
\end{array}
$$

II. A TÍPUSRENDSZEREK KÖVETKEZTETÉSEI

1. Az F_1 rendszer következtetései

$\Gamma \vdash wf$	Γ jól formált környezet
$\Gamma \vdash A$	Γ-ban az A jól formált típus
$\Gamma \vdash E{:}A$	Γ-ban az E kifejezés típusa A

2. Az F_2 rendszer következtetései:

$\Gamma \vdash wf$	Γ jól formált környezet
$\Gamma \vdash A$	Γ-ban az A jól formált típus
$\Gamma \vdash E{:}A$	Γ-ban az E kifejezés típusa A

A következtetések formailag megegyeznek az F_1 következtetéseivel. de az F_2 rendszerben a Γ típuskörnyezet bővebb, mint az F_1-ben, itt a típuskörnyezet típusváltozókat is tartalmaz.

III. A TÍPUSRENDSZEREK SZABÁLYAI

1. Az F_1 rendszer szabályai:

(a) *A környezetre vonatkozó szabályok:*

$$\frac{}{\emptyset \vdash wf} \quad [\textsc{Env } \emptyset]$$

$$\frac{\Gamma \vdash A \quad x \notin dom(\Gamma)}{\Gamma, x{:}A \vdash wf} \quad [\textsc{Env } x]$$

(b) *A típusra vonatkozó szabályok:*

$$\frac{\Gamma \vdash wf \quad A \in alaptípus}{\Gamma \vdash A} \quad [\textsc{Type Const}]$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \to B} \quad [\textsc{Type Arrow}]$$

(c) *Kifejezés típusára vonatkozó szabályok:*

$$\frac{\Gamma', x{:}A, \Gamma'' \vdash wf}{\Gamma', x{:}A, \Gamma'' \vdash x{:}A} \quad [\textsc{Val } x]$$

$$\frac{\Gamma, x{:}A \vdash E{:}B}{\Gamma \vdash \lambda x{:}A\,.\,E{:}A \to B} \quad [\textsc{Val Fun}]$$

$$\frac{\Gamma \vdash E{:}A \to B \quad \Gamma \vdash F{:}A}{\Gamma \vdash EF{:}B} \quad [\textsc{Val Appl}]$$

2. Az F_2 rendszer szabályai:

(a) *A környezetre vonatkozó szabályok:*

$$\frac{}{\emptyset \vdash \mathit{wf}} \quad [\textsc{Env } \emptyset]$$

$$\frac{\Gamma \vdash A \quad x \notin \mathit{dom}(\Gamma)}{\Gamma, x{:}A \vdash \mathit{wf}} \quad [\textsc{Env } x]$$

$$\frac{\Gamma \ \vdash \mathit{wf} \quad \alpha \notin \mathit{dom}(\Gamma)}{\Gamma, \alpha \vdash \mathit{wf}} \quad [\textsc{Env } \alpha]$$

(b) *A típusra vonatkozó szabályok:*

$$\frac{\Gamma', \alpha, \Gamma'' \vdash \ \mathit{wf}}{\Gamma', \alpha, \Gamma'' \vdash \alpha} \quad [\textsc{Type } \alpha]$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \to B} \quad [\textsc{Type Arrow}]$$

$$\frac{\Gamma, \alpha \vdash A}{\Gamma \vdash \forall \alpha \,.\, A} \quad [\textsc{Type Forall}]$$

(c) *Kifejezés típusára vonatkozó szabályok:*

$$\frac{\Gamma', x{:}A, \Gamma'' \vdash \mathit{wf}}{\Gamma', x{:}A, \Gamma'' \vdash x{:}A} \quad [\textsc{Val } x]$$

$$\frac{\Gamma, x{:}A \vdash E{:}B}{\Gamma \vdash \lambda x{:}A \,.\, E{:}A \to B} \quad [\textsc{Val Fun}]$$

$$\frac{\Gamma \vdash E{:}A \to B \quad \Gamma \vdash F{:}A}{\Gamma \vdash EF{:}B} \quad [\textsc{Val Appl}]$$

$$\frac{\Gamma, \alpha \vdash E{:}A}{\Gamma \vdash \Lambda \alpha \,.\, E{:}\forall \alpha \,.\, A} \quad [\textsc{Val Fun2}]$$

$$\frac{\Gamma \vdash E{:}\forall \alpha \,.\, A \quad \Gamma \vdash B}{\Gamma \vdash E[B]{:}A[\alpha := B]} \quad [\textsc{Val Appl2}]$$

2. Az $\boldsymbol{F_\omega}$ rendszer szabályai:

(a) *A környezetre vonatkozó szabályok:*

(b) *A fajtára vonatkozó szabályok:*

$$\frac{\Gamma \vdash K_1 : \Box \quad \Gamma \vdash K_2 : \Box}{\Gamma \vdash (K_1 \to K_2) : \Box} \quad [\textsc{Kind-Arrow}]$$

$$\frac{\Gamma, \alpha : K \vdash A : \star}{\Gamma \vdash \forall\alpha{:}K \,.\, A : \star} \quad [\textsc{Kind-Forall}]$$

$$\frac{\Gamma', \alpha : K, \Gamma'' \vdash \mathit{wf}}{\Gamma', \alpha : K, \Gamma'' \vdash \alpha : K} \quad [\textsc{Val-Kind}]$$

$$\frac{\Gamma, \alpha{:}K_1 \vdash A{:}K_2}{\Gamma \vdash \Lambda\alpha{:}K_1 \,.\, A : K_1 \to K_2} \quad [\textsc{Val-Fun-Kind}]$$

$$\frac{\Gamma \vdash A : K_1 \to K_2 \quad \Gamma \vdash B : K_2}{\Gamma \vdash AB : K_2} \quad [\textsc{Val-Appl-Kind}]$$

(c) *A típusra vonatkozó szabályok:*

$$\frac{\Gamma \vdash A{:}\star \quad \Gamma \vdash B{:}\star}{\Gamma \vdash (A \to B){:}\star} \quad [\textsc{Type Arrow}]$$

$$\frac{\Gamma \vdash A{:}\Box \quad \Gamma, \alpha{:}A \vdash B{:}\star}{\Gamma \vdash \forall\alpha{:}A \,.\, B{:}\star} \quad [\textsc{Type Forall}]$$

(c) *Kifejezés típusára vonatkozó szabályok:*

$$\frac{\Gamma', x{:}A, \Gamma'' \vdash \mathit{wf}}{\Gamma', x{:}A, \Gamma'' \vdash x{:}A} \quad [\textsc{Val } x]$$

$$\frac{\Gamma, x{:}A \vdash E{:}B}{\Gamma \vdash \lambda x{:}A \,.\, E{:}A \to B} \quad [\textsc{Val Fun}]$$

$$\frac{\Gamma \vdash E{:}A \to B \quad \Gamma \vdash F{:}A}{\Gamma \vdash EF{:}B} \quad [\textsc{Val Appl}]$$

$$\frac{\Gamma, \alpha \vdash E{:}A}{\Gamma \vdash \Lambda\alpha \,.\, E{:}\forall\alpha \,.\, A} \quad [\textsc{Val Fun2}]$$

$$\frac{\Gamma \vdash E{:}\forall\alpha \,.\, A \quad \Gamma \vdash B}{\Gamma \vdash E[B]{:}A[\alpha := B]} \quad [\textsc{Val Appl2}]$$

D. FÜGGELÉK

Definíciók, tételek jegyzéke

Névmutató

Tárgymutató

A, Á

B

C

CS

E, É

V

Z