EÖTVÖS LORÁND TUDOMÁNYEGYETEM
INFORMATIKAI KAR

CSÖRNYEI ZOLTÁN

FUNKCIONÁLIS PROGRAMNYELVEK IMPLEMENTÁCIÓJA

I. RÉSZ

A λ-KALKULUS[1]

BUDAPEST, 2003

[1] Részben az OTKA T037742 támogatásával

Jelölések

A könyvben az ábécék betűivel és a betűtípusokkal a következőket jelöljük:

$x, y, z, \ldots$	*λ-kalkulus és kombinátor logika változó*
$E, F, G, \ldots$	*λ-kalkulus és kombinátor logika kifejezés*
$\mathcal{C}, \mathcal{D}$	*λ-kalkulus kontextus*
$X, Y, Z, \ldots$	*λ-kalkulus kontextus változó*
$\alpha, \beta, \gamma, \ldots$	*típusos λ-kalkulus típusváltozó*
$A, B, C, \ldots$	*típusos λ-kalkulus típuskifejezés*
kurzív	*matematikai függvény*
$\mathsf{sans\ serif}$	*λ-kalkulus kifejezés neve*
sans serif	*kombinátor logika kifejezés neve*
$\equiv$	*ekvivalencia*
$=$	*egyenlőség*
$\to$	*típuskonstruktor*
$\rightarrowtriangle$	*redukció*
$\mapstochar\rightarrowtriangle$	*kombinátor logika optimalizálás*

Tartalomjegyzék

1. A λ-kalkulus **1**

1.1. A λ-kalkulus szintaktikája 2
1.1.1. λ-absztrakció 3
1.1.2. Az applikáció 4
1.1.3. Szabad és kötött változók 6
1.1.4. Helyettesítés 10
1.2. A λ-kalkulus operációs szemantikája 13
1.2.1. A β-konverzió 14
1.2.2. Az α-konverzió 17
1.2.3. Az α-konverzió elkerülése 20
1.2.4. Egyenlőség 24
1.2.5. A λ-kalkulus axiómái 27
1.2.6. Az η-konverzió és a kiterjeszthetőség 28
1.3. A λ-kifejezés normál formája 31
1.3.1. Redukálási stratégiák 33
1.4. Konstansok és konstans függvények 40
1.4.1. A logikai konstansok és műveletek 40
1.4.2. Pár, n-es és műveleteik 42
1.4.3. Listák és listákon értelmezett műveletek 45
1.4.4. A számkonstansok és aritmetikai műveletek 48
1.4.5. A δ-redukció 60
1.5. A rekurzió 61
1.5.1. A fixpont-kombinátor 63
1.5.2. A rekurzív függvények 68
1.5.3. A Russell-paradoxon 71
1.6. λ-definiálható függvények 73

1.6.1. Primitív rekurzív függvények 74
1.6.2. Teljes rekurzív függvények 77
1.6.3. Parciális rekurzív függvények 80
1.7. A megoldhatóság 82
1.7.1. Megoldható λ-kifejezés 82
1.7.2. A fej normál forma 87
1.7.3. A gyenge fej normál forma 91
1.7.4. λ-definiálható függvények 94
1.8. Konzisztens formális elmélet 99
1.8.1. A λ-kalkulus konzisztens 100
1.8.2. A megoldhatóság és a konzisztencia 103
1.8.3. Eldönthetetlen problémák 107

A. Definíciók, tételek jegyzéke **113**

Irodalom **115**

Névmutató **117**

Tárgymutató **119**

1. FEJEZET

A λ-kalkulus

A λ-kalkulus első definíciója és elméletének első eredményei Church-től származnak 1932-33-ból. Church célja az volt, hogy olyan kalkulust, azaz kifejezések definiálására és átalakítására szolgáló rendszert készítsen, amelylyel a függvények tulajdonságai, elsősorban kiszámíthatóságuk tulajdonságai jól vizsgálhatók.

A λ-kalkulus volt az első olyan „eszköz", amellyel a függvények kiszámíthatósága kezelhető volt. A klasszikus halmazelmélet, amely a függvényt argumentum–érték párok halmazának tekinti, nem alkalmas a rekurzió kezelésére, azaz benne egy függvény önmagával történő applikációja nem írható le.

A λ-kalkulus rendkívül egyszerű, és használhatóságát az adja, hogy a rekurzió benne a rekurzió explicit leírása nélkül leírható. A λ-kalkulus konzisztens, mint matematikai rendszer, nincs benne ellentmondás, nincsenek benne paradoxonok.

Kleene megmutatta, hogy minden kiszámítható függvény leírható λ-kalkulusban, és a λ-definiálható függvények pontosan a kiszámítható függvények. A Church–Rosser tételek pedig kimondják, hogy egy kifejezés értékének a meghatározásakor a számítás eredménye, ha létezik, független a kiszámításban alkalmazott stratégiától.

A λ-kalkulus az első funkcionális programnyelvnek tekinthető, annak ellenére, hogy kidolgozásának időpontjában még nem is voltak számítógépek. Ugyanakkor a λ-kalkulus egy olyan egyszerű funkcionális programnyelv, amelyre minden más "magasszintű" funkcionális nyelven írt program átalakítható. Ezért elegendő a λ-kalkulust interpretálni, egy λ-kalkulus-interpreter minden funkcionális nyelvű programot végrehajthat. Azonban a λ-kalkulus interpretálása már nehezebb feladat, a funkcionális nyelvek korszerű implementálására kifejezés- vagy gráf-redukciós módszereket használnak.

Minden funkcionális program egy kifejezésnek tekinthető, a program végrehajtása a kifejezés értékének meghatározását jelenti. Mint majd látni fogjuk, a funkcionális programokat alkotó kifejezések leírhatók a λ-kalkulus kifejezéseivel is. A λ-kalkulus kifejezéseit *λ-termeknek* vagy *λ-kifejezéseknek* nevezzük.

A kifejezések egyszerűbb leírására a funkcionális nyelvekben függvénydefiníciók adhatók meg, a kifejezésekben ezekre a függvényekre a definíciókban megadott függvénynevekkel lehet hivatkozni. A λ-kalkulusban erre nincs lehetőség, a λ-kalkulusban csak névtelen függvények definiálhatók.

A funkcionális program végrehajtása a λ-kifejezés kiértékelését, azaz a legegyszerűbb formára hozását jelenti. A λ-kalkulus ennek elvégzéséhez ad átalakítási szabályokat, azaz a λ-kalkulus a λ-kifejezések közötti olyan egyenlőségekből áll, amelyek levezethetők a konverziós szabályokból származtatott axiómákból.

1.1. A λ-kalkulus szintaktikája

Először megadunk egy *ábécének* nevezett halmazt, a λ-kalkulus jelkészletét. Ha ennek az ábécének a betűit egymás mellé írjuk, konkatenáljuk, akkor *szavakat* kapunk, ezek lesznek a λ-kalkulus kifejezései. A λ-kalkulus kifejezéseinek szintaktikáját úgy definiáljuk, hogy megmondjuk, mely szavak „helyesek”. Ezután az operációs szemantikával foglalkozunk, és csak ezután adjuk meg a λ-kalkulus pontos definícióját.

1.1.1. Definíció. Az egyszerű típusnélküli λ-kifejezések:

Tegyük fel, hogy adott egy nem feltétlenül véges, egymástól páronként különböző változókat (szimbólumokat), a λ jelet, a pontot, valamint a nyitó- és csukózárójeleket tartalmazó halmaz. Ezen a halmazon, mint ábécén értelmezett λ-kifejezések a következő szavak:

$$
\begin{array}{lll}
<\lambda\text{-}\mathit{kifejez\acute{e}s}> & ::= & <\mathit{v\acute{a}ltoz\acute{o}}> \\
 & | & <\lambda\text{-}\mathit{absztrakci\acute{o}}> \\
 & | & <\mathit{applik\acute{a}ci\acute{o}}> \\
<\lambda\text{-}\mathit{absztrakci\acute{o}}> & ::= & (\lambda \ <\mathit{v\acute{a}ltoz\acute{o}}> \ . <\lambda\text{-}\mathit{kifejez\acute{e}s}> \ ) \\
<\mathit{applik\acute{a}ci\acute{o}}> & ::= & (<\lambda\text{-}\mathit{kifejez\acute{e}s}> <\lambda\text{-}\mathit{kifejez\acute{e}s}>)
\end{array}
$$

A λ-kalkulusban, ellentétben a funkcionális nyelvekkel, nincs típusfogalom, és az „egyszerű” jelző arra utal, hogy nincsenek benne sem konstansok, sem konstansokon értelmezett függvények.

A továbbiakban a λ-kifejezéseket nagybetűkkel, a változókat kisbetűkkel jelöljük, és a λ-kifejezésekben a legkülső zárójelpárt gyakran elhagyjuk. A λ-kifejezések halmazát Λ-val jelöljük.

Két λ-kifejezés szintaktikus *azonosságára* az $\equiv$ jelet használjuk, és azt mondjuk, hogy E „szintaktikusan azonos”, vagy röviden „azonos” F-fel, azaz $E \equiv F$, ha az E és F λ-kifejezések pontosan megegyeznek. Természetesen feltesszük, hogy ha $EF \equiv GH$, akkor $E \equiv G$ és $F \equiv H$, és ha $\lambda x.E \equiv \lambda y.F$, akkor $x \equiv y$ és $E \equiv F$.

Az $\equiv$ reláció *equivalencia reláció*, azaz *reflexív*, *szimmetrikus* és *tranzitív*.

Az azonosságot arra is használjuk, hogy λ-kifejezéseknek nevet adjunk, és ezzel a λ-kifejezések leírását lerövidítsük. Ezeket az emlékeztető neveket, *mnemonikokat* Sans Serif betűtípussal írjuk.

A λ-kifejezések azonos átalakításával részletesen az 1.1.4. pontban foglalkozunk.

1.1.1. λ-absztrakció

A λ-kalkulusban minden függvény anonym, és függvények csak λ-absztrakcióval adhatók meg. Az 1.1.1. definícióban szereplő λ jelet λ-absztraktornak nevezzük, és a definícióból az is látható, hogy a λ-absztraktor unáris operátor. A λ-absztrakció egy olyan λ-kifejezés, amely egy névtelen függvény definiálására szolgál:

1.1.2. Definíció. λ-absztrakció:

Ha E egy λ-kifejezés és x egy változó, akkor a

$\lambda x.E$

λ-kifejezést λ-absztrakciónak, az x változót a λ-absztrakció változójának vagy formális paraméterének, az E λ-kifejezést a λ-absztrakció törzsének nevezzük.

Az E kifejezés a λ-absztrakció x változójának a *hatásköre*, és azt mondjuk, hogy a hatáskörbe tartozó x változót a λ-absztrakció x változója köti. A λ-absztrakció *jobbasszociatív*, például, ha az E kifejezés is egy $\lambda y.F$ λ-absztrakció, akkor $\lambda x.(\lambda y.F)$ röviden így is írható:

$$\lambda x.(\lambda y.F) \equiv \lambda x.\lambda y.F \equiv \lambda xy.F,$$

azaz a redundáns zárójelpárok elhagyhatók, és a λ-absztrakció lambdái összevonhatók. Ha $\vec{x}$-lal jelöljük az $x_1x_2\ldots x_n$ sorozatot ($n \geq 1$), akkor

$$\lambda x_1x_2\ldots x_n.E \equiv \lambda\vec{x}.E.$$

1.1.2. Az applikáció

Mint az 1.1.1. definícióból látható, az applikáció bináris operátor, a jele gyakran a $\cdot$, vagy mint az 1.1.3. pontban látni fogjuk, a λ-kifejezés gráfjában az applikációt a @ jel jelöli. Ebben a jegyzetben az applikációt a λ-kifejezések egymás mellé írásával, vagy ha a jobb olvashatóság miatt szükséges, akkor a λ-kifejezések közé írt szóköz karakterrel jelöljük.

1.1.3. Definíció. Az applikáció:

Ha E és F λ-kifejezések, akkor az

EF

λ-kifejezést applikációnak, ha E egy λ-absztrakció, akkor az EF-t függvényapplikációnak nevezzük. Az F az E λ-kifejezés aktuális paramétere.

Az applikáció *balasszociatív*, így például

$(xy)z \equiv xyz,$

azaz a redundáns zárójelpárok elhagyhatók. Hasonlóan a változókból álló vektorhoz, ha $\vec{E}$-lal jelöljük az $E_1E_2\ldots E_n$ sorozatot ($n \geq 1$), akkor

$\lambda x.E_1E_2\ldots E_n \equiv \lambda x.\vec{E}.$

Megjegyezzük, hogy a $\lambda\vec{x}.\vec{x}$ λ-kifejezésben a két vektor nem azonos, hiszen a λ-absztrakció változóira a jobbasszociatívitás, a törzsben levő változókra a balasszociatívitás érvényes.

Az applikációnak nagyobb a *precedenciája*, mint a λ-absztrakciónak, így a redundáns zárójelpárok a λ-absztrakció törzsében is elhagyhatók:

$\lambda x.(yz) \equiv \lambda x.yz,$
$\lambda x.((\lambda y.E)F) \equiv \lambda x.(\lambda y.E)F.$

1.1.4. Példa. *(Három gyakran használt λ-kifejezés)*
Az I „identitás”, a K „konstans” és a S „kompozíció” függvény:

$\mathsf{I} \equiv \lambda x.x,$
$\mathsf{K} \equiv \lambda xy.x,$
$\mathsf{S} \equiv \lambda xyz.xz(yz).$ □

Megjegyezzük, hogy a λ-kalkulusban minden függvénynek csak egy formális paramétere és így csak egy aktuális paramétere lehet, a többparaméteres matematikai függvények egyváltozós *magasabbrendű függvények* bevezetésével alakíthatók át egyváltozós függvények applikációinak sorozatává. Ezt az átalakítást *currying*-nak, „karrizásnak” nevezzük.

1.1.5. Példa. *(Currying)*
Ebben a példában a hagyományos matematikai jelöléseket használva, alakítsuk át az $add(x, y) = x + y$ kétváltozós függvényt. Legyen $f(x)$ egy olyan magasabbrendű függvény, amely az x argumentumára az $f(x) = add_x$ függvényt adja „értékül”, ahol az add_x egy olyan egyváltozós függvény, amely az argumentumát x-szel növeli. Így

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

azaz a kétváltozós *add* függvényt két egyváltozós függvénnyel írtuk le. □

1.1.3. Szabad és kötött változók

Skatulyázott, azonos változójú λ-absztrakciók törzsében a λ-absztrakciók változója mindkét λ-absztrakció hatáskörében benne lehet:

1.1.6. Példa. *(Azonos változójú λ-absztrakciók)*

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

$\lambda xx.x \;\equiv\; \lambda x.(\lambda x.x)$. □

Hasonlóan a procedúrális programnyelvekben használt „a deklaráció láthatósága" fogalomhoz, a λ-kalkulusban is beszélhetünk arról, hogy a törzsben levő változó melyik λ-absztrakcióhoz tartozik, azaz melyik λ-absztrakció köti.

1.1.7. 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 $\lambda y.E$-ben, ha $x \neq y$ és az x szabad E-ben.*

1.1.8. Definíció. A kötött változó:

1. *Az x kötött $\lambda y.E$-ben, ha $x = y$ és az x szabad E-ben, vagy ha az x kötött E-ben,*
2. *az x kötött EF-ben, ha kötött E-ben, vagy ha kötött F-ben.*

Ha a $\lambda x.E$ λ-absztrakcióban az x változó az E szabad változója, akkor azt mondjuk, hogy ezt a szabad x változót a λ-absztrakció köti.

A definícióból látható, hogy ha egy változó egy λ-kifejezés belsejében szabad változó, akkor az egész λ-kifejezésre a változó kötötté válhat, de egy belül kötött változó a külső λ-kifejezésekben sohasem válhat szabaddá.

Ha $FV(E)$ jelöli az E kifejezés szabad változóinak halmazát, akkor az 1.1.7. definíció alapján

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

1.1.9. Példa. *(Az 1.1.6. példa folytatása)*
A definíciók szerint tehát a fenti példa első λ-kifejezésében a törzs első x változója a belső λ-absztrakcióban, a belső applikációban és a külső λ-absztrakcióban is kötött változó, míg a második x változó a belső applikációban szabad, a külső λ-absztrakcióban kötött változó.

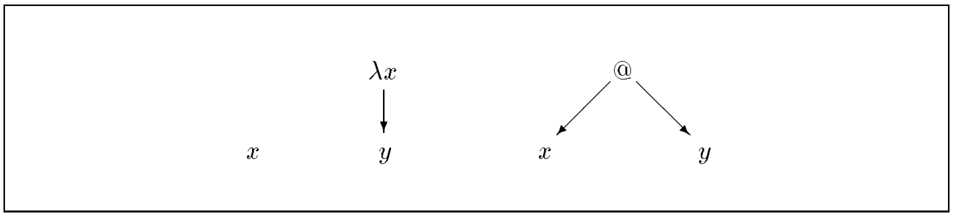
A második λ-kifejezésben a törzs x változója mind a belső, mind a külső λ-absztrakcióban kötött változó, de az x változót csak a belső λ-absztrakció köti. □

1.1.10. Példa. *(Kötött és szabad változók)*
$\lambda x.(\lambda x.xy)$,
$\lambda y.(\lambda x.xy)$,
$\lambda x.(\lambda y.xy)$,
$\lambda y.(\lambda y.xy)$.
Ezekben a λ-kifejezésekben az x változó a belső λ-absztrakciókban rendre kötött, kötött, szabad és szabad változó, míg a külső λ-absztrakciókban rendre kötött, kötött, kötött és szabad változó. □

Az, hogy egy változó kötött vagy szabad, a *λ-kifejezés gráfjából* is könnyen meghatározható.

Rendeljük hozzá az x, $\lambda x.y$ és az xy λ-kifejezésekhez az 1.1. ábrán látható gráfokat, a @ szimbólum legyen az applikáció jele. Ezekből az elemekből egy tetszőleges λ-kifejezés gráfja felépíthető.

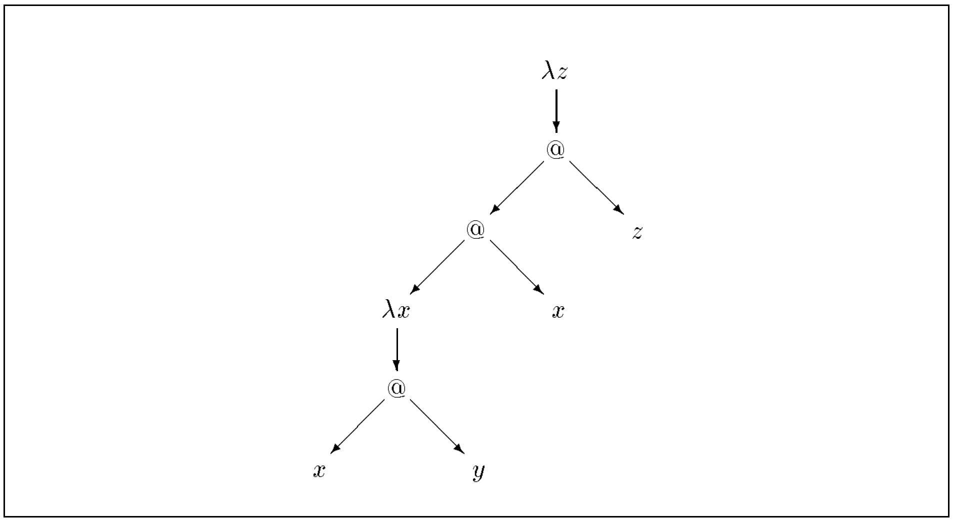
A gráf felépítéséből látható, hogy a fák levelein levő változók λ-absztrakciók törzsének vagy applikációknak a változói. Egy x változó kötött változó, ha a λ-kifejezés gráfjában az x levélből kiindulva, felfelé, a gráf gyökérpontja felé haladva egy λx gráfponthoz jutunk. Az első λx-szel címkézett gráfponthoz tartozó λ-absztrakció köti az x változót. Ellenkező



1.1. ábra. Az x, $\lambda x.y$ és az xy λ-kifejezések gráfja

esetben, azaz ha úgy feljutunk a gráf csúcsába, hogy nem megyünk át λx gráfponton, az x változó a λ-kifejezésben szabad változó.

1.1.11. Példa. *(Kötött és szabad változók meghatározása gráfból)* Határozzuk meg, hogy a $\lambda z.(\lambda x.xy)xz$ λ-kifejezésben melyik változó kötött, melyik szabad. A λ-kifejezés gráfja az 1.2. ábrán látható.



1.2. ábra. A $\lambda z.(\lambda x.xy)xz$ λ-kifejezés gráfja

Az ábráról leolvasható, hogy az első x és a z változó kötött, az y és a második x pedig szabad változó. $\square$

Egy λ-kifejezés gráfjából a λ-kifejezés részkifejezései is könnyen meghatározhatók. Egy λ-kifejezés *részkifejezését* a következőképpen definiáljuk:

1.1.12. Definíció. Egy λ-kifejezés részkifejezése:

Az E, F és G λ-kifejezésre és az x változóra

- *az E önmagának részkifejezése,*
- *ha F az E részkifejezése, akkor F*
  - *a $\lambda x.E$-nek is részkifejezése,*
  - *az EG-nek és a GE-nek is részkifejezése.*

Látható, hogy a részkifejezések a λ-kifejezés gráfjában a „részfáknak” felelnek meg.

1.1.13. Példa. *(Részkifejezések)*
A $\lambda x.yx(\lambda xy.yx)$ λ-kifejezésben balról jobbra haladva a következő részkifejezések vannak:

$\lambda x.yx(\lambda xy.yx)$, y, yx, $yx(\lambda xy.yx)$, x, $\lambda xy.yx$, $\lambda y.yx$, y, yx, x.

Látható, hogy ugyanaz a λ-kifejezés többször is előfordulhat részkifejezésként. Az $x(\lambda xy.yx)$ és a $\lambda xy.y$, vagy például a $\lambda y.y$ nem részkifejezések. □

1.1.14. Definíció. Zárt λ-kifejezés:

Ha egy λ-kifejezésben nincs szabad változó, akkor a λ-kifejezést zártnak nevezzük.

A zárt λ-kifejezéseket *kombinátoroknak* nevezzük, és a zárt λ-kifejezések halmazát Λ^0-val jelöljük.

1.1.15. Példa. *(Zárt λ-kifejezések)*
Az 1.1.4. példában szereplő I, K és S λ-kifejezések zárt λ-kifejezések, azaz kombinátorok voltak. □

Megjegyezzük, hogy a helyes, kiértékelhető funkcionális programok zárt λ-kifejezések, mivel egy nem zárt λ-kifejezés kiértékelésekor a szabad változók biztosan nem kapnak értéket. A zártság tehát a kiértékelhetőség szükséges feltétele.

1.1.16. Definíció. λ-kifejezés lezárása:

Ha $\{x_1, x_2, \ldots, x_n\} = FV(E)$, akkor $\lambda\vec{x}.E$-t az E egy lezárásának nevezzük.

Egy λ-kifejezésnek több lezárása is lehet, az E különböző lezárásai az $FV(E)$ halmaz elemeinek, azaz az absztrakció változóinak permutációival adhatók meg.

1.1.17. Példa. *(Lezárás)*
A $\lambda x.xyz$-nek két lezárása van: $\lambda yzx.xyz$ és $\lambda zyx.xyz$. □

1.1.4. Helyettesítés

Ha az E λ-kifejezésben az x *szabad változókat* az F λ-kifejezéssel helyettesítjük, akkor a helyettesítéssel kapott λ-kifejezést $E[x := F]$-fel jelöljük, ezt gyakran $[F/x]E$-vel, $[x \mapsto F]E$-vel, vagy $E[F/x]$-szel is jelölik. Megjegyezzük, hogy az $E[x := F]$ nem λ-kifejezés, hanem ezt a karaktersorozatot csak egy λ-kifejezés jelölésére használjuk.

Most azt vizsgáljuk meg, hogy az $E[x := F]$ milyen λ-kifejezéssel azonos.

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

Az 1.1.18. definíció 3. pontjának első része azt mondja ki, hogy egy λ-absztrakció változója nem helyettesíthető, a második rész szerint egy λ-

absztrakció törzsében levő y szabad változó csak akkor helyettesíthető G-vel, ha G-ben nincs szabad x változó, mert hiszen ez az x a helyettesítés után kötötté válna. Ha ezt megengednénk, akkor a λ-kalkulus nem lenne *konzisztens*, mint azt a következő példa mutatja:

1.1.19. Példa. *(A szabad változó nem válhat kötötté)*
Tekintsük a $\lambda x.y$ λ-kifejezést, ez egy olyan konstans függvény, amely mindig az y-t adja eredményül. Ekkor, ha az 1.1.18. definíció 3. pontjában az $x \notin FV(G)$ feltételt nem követelnénk meg, a

$\lambda x.y[y := x] \equiv \lambda x.x$

lenne, azaz a helyettesítés után a konstans függvény az identitás függvénnyé válna. $\square$

Megjegyezzük, hogy az 1.1.18. definíció 3. pontjában ha $x \not\equiv y$ és $x \in FV(G)$, akkor a helyettesítés a λ-absztrakció változójának megváltoztatásával megoldható. Erre majd az 1.2.2. pontban még visszatérünk, az 1.1.19. példában levő helyettesítésre a helyes megoldást az 1.2.14. példában adjuk majd meg.

1.1.20. Példa. *(Kifejezések helyettesítése)*
$(\lambda x.y)[x := \lambda x.y] \equiv \lambda x.y,$
$(\lambda x.y)[y := \lambda x.y] \equiv \lambda x.\lambda x.y \equiv \lambda xx.y,$
$(\lambda x.y)[y := \lambda y.x] \equiv \lambda x.y$ *(lásd az 1.2.14. példát)*,
$(\lambda x.y)[y := \lambda x.x] \equiv \lambda x.\lambda x.x \equiv \lambda xx.x,$
$(\lambda x.y)[y := \lambda y.y] \equiv \lambda x.\lambda y.y \equiv \lambda xy.y.$ $\square$

A helyettesítés *balasszociatív*, így a redundáns zárójeleket elhagyva

$(E[x := F])[y := G] \equiv E[x := F][y := G].$

A helyettesítésekre nyilvánvalóan igazak a következő egyszerű állítások:

1.1.21. Lemma. (Egyszerű helyettesítések)

‖ *1.* $E[x := x] \equiv E,$

2. $E[x := F] \equiv E$, *ha* $x \notin FV(E)$,
3. $(E[x := y])[y := x] \equiv E$, *ha* $y \notin FV(E)$,
4. $(E[x := y])[y := F] \equiv E[x := F]$, *ha* $y \notin FV(E)$,
5. $(E[x := F])[x := G] \equiv E[x := F[x := G]]$.

Bizonyítás. Az első két állítás teljesen nyilvánvaló. A harmadik állítás a negyedik és az első állításból következik. A negyedik állítás triviális, a feltétel azért szükséges, hogy a baloldal második lépésében csak azokat az y változókat cseréljük F-re, amelyek az első lépésben kerültek a λ-kifejezésbe. Az ötödik állítás is egyszerűen belátható, hiszen a baloldal második lépésében csak azokat az x változókat helyettesítjük, amelyek az F-ből kerültek a kifejezésbe. □

1.1.22. Lemma. (Az $E[x := F]$ szabad változói)
Ha $x \in FV(E)$, *akkor*

$$FV(E[x := F]) = (FV(E) \setminus \{x\}) \cup FV(F).$$

Bizonyítás. Az állítás a helyettesítés definíciójának a következménye. □

1.1.23. Lemma. (A helyettesítési lemma)
Ha $x \not\equiv y$ *és* $x \notin FV(G)$, *akkor*

$$E[x := F][y := G] \;\equiv\; E[y := G][x := F[y := G]].$$

Bizonyítás. A bizonyítás ez E szerkezete szerinti indukcióval történik.

1. eset: Ha $E \equiv x$, akkor az 1.1.18. definíció 1. pontja alapján

$$E[x := F][y := G] \equiv$$
$$x[x := F][y := G] \equiv$$
$$F[y := G],$$

a jobboldal pedig az 1.1.21. lemma 2. állítása és az 1.1.18. definíció 1. pontja alapján

$$E[y := G][x := F[y := G]] \equiv$$

$x[y := G][x := F[y := G]] \equiv$
$x[x := F[y := G]] \equiv$
$F[y := G]$.

2. eset: Ha $E \equiv y$, akkor a bizonyítás az 1.1.21. lemma állításait és a 1.1.18. definíciót felhasználva az 1. esethez hasonlóan végezhető el.

3. eset: Ha $E \equiv z \not\equiv x$ és $z \not\equiv y$, akkor nyilván mindkét oldal z-vel egyenlő.

4. eset: Ha $E \equiv E_1E_2$, akkor az 1.1.18. definíció és az indukciós feltétel alapján

$(E_1E_2)[x := F][y := G] \equiv$
$E_1[x := F][y := G]E_2[x := F][y := G] \equiv$
$E_1[y := G][x := F[y := G]]E_2[y := G][x := F[y := G]] \equiv$
$(E_1[y := G]E_2[y := G])[x := F[y := G]] \equiv$
$E_1E_2[y := G][x := F[y := G]]$.

5. eset: Legyen $E \equiv \lambda z.E_1$, és feltehetjük (lásd az 1.2.2. pontot), hogy $z \not\equiv x$ és $z \not\equiv y$, $z \notin FV(F)$ és $z \notin FV(G)$. Ekkor az 1.1.18. definíció és az indukciós feltétel alapján

$(\lambda z.E_1)[x := F][y := G] \equiv$
$\lambda z.E_1[x := F][y := G] \equiv$
$\lambda z.E_1[y := G][x := F[y := G]] \equiv$
$(\lambda z.E_1)[y := G][x := F[y := G]]$. □

1.2. A λ-kalkulus operációs szemantikája

A λ-kalkulus szemantikáját *konverziós szabályokkal* írjuk le, amelyek megadják, hogy egy λ-kifejezést hogyan lehet egy másik λ-kifejezésbe transzformálni. A konverziós szabályok *reflexív*, *szimmetrikus* és *tranzitív* tulajdonsága biztosítja azt, hogy a konverziós szabályokkal átalakított λ-kifejezés az eredeti λ-kifejezésre visszakonvertálható.

1.2.1. A β-konverzió

A λ-absztrakció egy függvényt jelöl, és egy λ-absztrakcióból és egy λ-kifejezésből álló függvényapplikáció második tagja a függvény aktuális paramétere. A β-redukció megadja, hogy egy függvényt az aktuális paraméterére alkalmazva milyen eredményt kapunk.

1.2.1. Definíció. A β-redukció:

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

$(\lambda x.E)F \rightarrow_\beta E[x := F]$.

Ha a fenti definícióban szereplő feltétel nem teljesül, azaz az F szabad változói kötötté válnak, akkor a β-redukció nem hajtható végre.

A definíció azt mondja ki, hogy ha az F szabad változói közül egyik sem lesz kötött az E-ben, akkor a függvény az aktuális paraméterével mindig egy *redukálható kifejezést*, *redexet* ad. A redukció eredménye a λ-absztrakció törzse, amelyben a λ-absztrakció változóját a λ-absztrakcióval kötött minden előfordulási helyén „szövegesen" az aktuális paraméterrel kell helyettesíteni.

A továbbiakban, ha a jobb olvashatóság érdekében szükséges, a β-redukcióval redukálandó redexet aláhúzással jelöljük.

Ha a β-redukciót a „fordított irányban" alkalmazzuk, azaz egy λ-kifejezésből egy olyan applikációt írunk fel, amelyben az első tag egy λ-absztrakció, akkor ezt az átalakítást *β-absztrakciónak* nevezzük. A β-redukció és a β-absztrakció közös neve a *β-konverzió*, a β-absztrakciót $\leftarrow_\beta$ -val, a β-konverziót $\leftrightarrow_\beta$ -val jelöljük.

A β-konverziót tetszőleges λ-kifejezésre is definiálhatjuk, azt mondjuk, hogy $E \leftrightarrow_\beta F$, ha az E egy részkifejezésére a β-konverziót alkalmazva az F-t kapjuk meg:

1.2.2. (Leibniz-szabály:)

Ha $E_1 \leftrightarrow_\beta F_1$, E_1 az E λ-kifejezés egy részkifejezése, és F az E-től csak abban különbözik, hogy benne az E_1 részkifejezés helyén az F_1 λ-kifejezés van, akkor $E \leftrightarrow_\beta F$.

A Leibniz-szabály speciális esetei a következők:

1.2.3. Következmény.

Ha $E \twoheadleftarrow\!\!\twoheadrightarrow_\beta F$*, akkor egy tetszőleges* G *λ-kifejezésre*

1. $GE \twoheadleftarrow\!\!\twoheadrightarrow_\beta GF$,
2. $EG \twoheadleftarrow\!\!\twoheadrightarrow_\beta FG$,
3. $\lambda x.E \twoheadleftarrow\!\!\twoheadrightarrow_\beta \lambda x.F$.

Az utolsó eset egy speciális nevet is kapott:

1.2.4. Definíció. ξ-szabály:

A ξ-szabály a következő:

ha $E \twoheadleftarrow\!\!\twoheadrightarrow_\beta F$*, akkor* $\lambda x.E \twoheadleftarrow\!\!\twoheadrightarrow_\beta \lambda x.F$.

1.2.5. Példa. *(β-redukciók)*

$\underline{(\lambda x.x)y} \twoheadrightarrow_\beta y$,

$\underline{(\lambda x.y)z} \twoheadrightarrow_\beta y$,

$\underline{(\lambda y.(\lambda x.yx))u}y \twoheadrightarrow_\beta$
$\underline{(\lambda x.ux)y} \twoheadrightarrow_\beta uy$. □

1.2.6. Példa. *(Az argumentum kifejezés is lehet)*

$\underline{(\lambda f.fu)(\lambda x.xy)} \twoheadrightarrow_\beta$
$\underline{(\lambda x.xy)u} \twoheadrightarrow_\beta uy$. □

1.2.7. Példa. *(A β-redukció alkalmazásakor csak a λ-absztrakció szabad változóit szabad helyettesíteni az argumentummal)*

$\underline{(\lambda x.(\lambda x.xy)x)u} \twoheadrightarrow_\beta$
$\underline{(\lambda x.xy)u} \twoheadrightarrow_\beta uy$. □

1.2.8. Lemma. (A β-redukció és a szabad változók)

Ha $E \twoheadrightarrow_\beta F$ *és* $x \notin FV(E)$*, akkor* $x \notin FV(F)$.

Bizonyítás. A lemma a β-konverzió definíciójából következik. □

1.2.9. Lemma. **(β-redukciók és helyettesítések)**

Ha $E \twoheadrightarrow_\beta F$, *akkor tetszőleges* G *-re*

1. $G[x := E] \twoheadrightarrow_\beta G[x := F]$,

2. $E[x := G] \twoheadrightarrow_\beta F[x := G]$.

Bizonyítás. Az első állítás nyilvánvaló, a G-ben levő E részkifejezésekre kell elvégezni az $E \twoheadrightarrow_\beta F$ redukciót. A második állításra feltehetjük, hogy

$$E \equiv (\lambda y.H)I,$$
$$F \equiv H[y := I],$$

és $x \not\equiv y$, $y \notin FV(G)$. Ekkor a

$$((\lambda y.H)I)[x := G] \twoheadrightarrow_\beta H[y := I][x := G]$$

redukciót kell igazolni. A helyettesítés 1.1.18. definíciója alapján

$$((\lambda y.H)I)[x := G] \equiv$$
$$((\lambda y.H)[x := G])\ I[x := G] \equiv$$
$$(\lambda y.H[x := G])\ I[x := G].$$

A β-redukciót végrehajtva, majd az 1.1.23. lemma alapján

$$H[x := G][y := I[x := G]] \equiv$$
$$H[y := I[x := G]]\ [x := G[y := I[x := G]]],$$

de mivel $y \notin FV(G)$,

$$[x := G[y := I[x := G]]] \equiv [x := G],$$

azaz

$$H[y := I[x := G]][x := G] \equiv$$
$$H[y := I][x := G],$$

és így a második állítást is igazoltuk. □

1.2.2. Az α-konverzió

A β-redukciót nem lehet végrehajtani, ha a paraméter szabad változója kötötté válik.

1.2.10. Példa. *(A β-redukció nem alkalmazható)*
$\underline{(\lambda xy.xy)y} \;\not\rhd_\beta\; \lambda y.yy,$
$\underline{(\lambda xy.x)y} \;\not\rhd_\beta\; \lambda y.y.$

A jobboldali λ-kifejezésekben a helyettesítéssel kapott y változót a λ-absztrakció y változója köti. □

A problémát az okozza, hogy különböző változóknak azonos nevük van, ezt a változók *névkonfliktusának* nevezzük. Ez a konfliktus az α-konverzióval oldható meg, az α-konverzióban a λ-absztrakció változóját fogjuk helyettesíteni.

1.2.11. Definíció. Az α-konverzió:
Ha az E-ben nincs y szabad változó, azaz $y \notin FV(E)$, akkor

$\lambda x.E \;\lhd\!\rhd_\alpha\; \lambda y.E[x := y]$.

A definícióban szereplő feltétel azért szükséges, hogy az E szabad y változója ne váljon az α-konverzió végrehajtása után kötött változóvá.

Az α-konverzió alkalmazásához tehát egy olyan változót kell keresnünk, amelyik vagy nem szerepel E-ben, azaz E-nek nem részkifejezése, vagy ha az E-ben szerepel, akkor nem lehet az E-nek szabad változója. Feltesszük, hogy a λ-kalkulus változóinak halmazát úgy adjuk meg, hogy mindig találjunk ilyen változót, ezt a *változókra vonatkozó konvenciónak* nevezzük.

Látható, hogy az α-konverzió egy olyan speciális helyettesítés (lásd az 1.1.4. pontot), amelyben egy λ-absztrakció változója és a λ-absztrakció törzsében a változóval kötött szimbólumok helyettesítődnek. Mivel az α-konverzió végrehajtása után az eredeti λ-kifejezéssel szintaktikusan azonos λ-kifejezést kapunk, azaz $E \;\lhd\!\rhd_\alpha\; F$-ből az $E \equiv F$ következik, és az α-konverziót gyakran $\equiv_\alpha$-val is jelölik.

Megjegyezzük, hogy mivel az α-konverzió csak szintaktikus módosítást végez és a λ-kifejezés stuktúráját nem változtatja meg, az α-konverzió *nem* *tekinthető explicit redukciós lépésnek.*

Hasonlóan a β-konverzióhoz, az $E \twoheadleftarrow\!\!\!\!\twoheadrightarrow_\alpha F$ azt jelenti, hogy az E egy részkifejezésére az α-konverziót alkalmazva az F-t kapjuk eredményül.

1.2.12. Példa. *(Az 1.2.10. Példa folytatása)*

$(\lambda xy.xy)y \twoheadleftarrow\!\!\!\!\twoheadrightarrow_\alpha \underline{(\lambda xz.xz)y} \twoheadrightarrow_\beta \lambda z.yz,$

$(\lambda xy.x)y \twoheadleftarrow\!\!\!\!\twoheadrightarrow_\alpha \underline{(\lambda xz.x)y} \twoheadrightarrow_\beta \lambda z.y.$ □

Az α-konverzió felhasználásával a helyettesítés 1.1.18. definíciójának 3. pontja így módosítható:

1.2.13. Definíció. Az 1.1.18. Definíció 3. pontja:

$$\text{3. } (\lambda x.E)[y := G] \equiv \begin{cases} \lambda x.E, & \text{ha } x \equiv y, \\ \lambda x.E[y := G], & \text{ha } x \not\equiv y \text{ és } x \notin FV(G), \\ \leftrightarrow_\alpha (\lambda z.E[x := z])[y := G] \equiv \lambda z.E[x := z][y := G], & \\ & \text{ha } x \not\equiv y \text{ és } x \in FV(G). \end{cases}$$

1.2.14. Példa. *(Helyettesítések α-konverzió után)*

Az 1.1.19. példa helyes megoldása:

$\lambda x.y[y := x] \twoheadleftarrow\!\!\!\!\twoheadrightarrow_\alpha \lambda z.y[y := x] \equiv \lambda z.x.$

Az 1.1.20. példa 3. sorában levő λ-kifejezésre:

$\lambda x.y[y := \lambda y.x] \twoheadleftarrow\!\!\!\!\twoheadrightarrow_\alpha \lambda z.y[y := \lambda y.x] \equiv \lambda z.(\lambda y.x) \equiv \lambda zy.x.$ □

Névkonfliktus β-redukciók egymásutáni alkalmazásakor akkor is előfordulhat, ha az eredeti λ-kifejezésben névkonfliktus nincs:

1.2.15. Példa. *(Névkonfliktus β-redukciókból)*

Legyen $T \equiv \lambda fx.f(fx)$. Ekkor

$TT \equiv \underline{(\lambda fx.f(fx))T} \twoheadrightarrow_\beta$

$\lambda x.T(Tx) \equiv \lambda x.T(\underline{(\lambda fx.f(fx))x}).$

A β-redukció most nem alkalmazható, mert akkor az x aktuális paraméter kötötté válna. Az α-konverziót alkalmazva:

$\lambda fx.f(fx) \twoheadleftarrow\!\!\!\!\twoheadrightarrow_\alpha \lambda fv.f(fv),$

és így

$\lambda x.T(\underline{(\lambda fv.f(fv))x}) \twoheadrightarrow_\beta$
$\lambda x.T(\lambda v.x(xv)) \equiv$
$\lambda x.\underline{(\lambda fx.f(fx))(\lambda v.x(xv))} \leftrightarrow_\alpha$
$\lambda x.\underline{(\lambda fw.f(fw))(\lambda v.x(xv))} \twoheadrightarrow_\beta$
$\lambda x.(\lambda w.(\lambda v.x(xv))(\underline{(\lambda v.x(xv))w})) \twoheadrightarrow_\beta$
$\lambda x.(\lambda w.\underline{(\lambda v.x(xv))(x(xw))}) \twoheadrightarrow_\beta$
$\lambda x.(\lambda w.(x(x(x(xw))))) \leftrightarrow_\alpha$
$\lambda f.(\lambda w.(f(f(f(fw))))) \leftrightarrow_\alpha$
$\lambda f.(\lambda x.(f(f(f(fx))))) \equiv$
$\lambda fx.f(f(f(fx))).$ □

1.2.16. Lemma. (Az α-konverzió és a szabad változók)

Ha $E \leftrightarrow_\alpha F$*, akkor* $FV(E) = FV(F)$.

Bizonyítás. Az állítás az α-konverzió definíciójából és az 1.1.21. lemmából következik. □

Ha az 1.1.21. lemma 2.–4. pontjaiban az $x \notin FV(E)$ és az $y \notin FV(E)$ feltételeket nem követeljük meg, akkor a lemma állításai nem azonosságok, hanem α-konverziók lesznek:

1.2.17. Lemma. (Egyszerű α-konverziók)

2. $E[x := F] \leftrightarrow_\alpha E,$
3. $(E[x := y])[y := x] \leftrightarrow_\alpha E,$
4. $(E[x := y])[y := F] \leftrightarrow_\alpha E[x := F].$

Bizonyítás. Az állítások az E szerkezetére vonatkozó indukcióval könnyen bizonyíthatók. □

1.2.18. Lemma. (α-konverziók és helyettesítések)

Ha $E \leftrightarrow_\alpha E'$ *és* $F \leftrightarrow_\alpha F'$*, akkor* $E[x := F] \leftrightarrow_\alpha E'[x := F']$.

Bizonyítás. A lemma állítása a definíciók triviális következménye. □

1.2.3. Az α-konverzió elkerülése

Az 1.2.2. pontban láttuk, hogy egy β-redukció előtt meg kell vizsgálni, hogy a redukció végrehajtása után nem lép-e fel névkonfliktus, mert akkor a redukció előtt egy α-konverziót kell végrehajtani. Bár az α-konverzió egy egyszerű műveletnek tűnik, implementálása nem könnyű, és gyakori végrehajtása a λ-kalkulus normál formájának előállítását jelentősen lelassíthatja.

Az α-konverzió elkerülésére két módszert ismertetünk, az első de Bruijntól származik, a másik módszert egy későbbi fejezetben tárgyaljuk.

Mivel a névkonfliktus oka az, hogy különböző változókat azonos névvel jelölhetünk, ennek a lehetőségnek a megszüntetése az α-konverziót is feleslegessé teszi.

Egy λ-kifejezésben helyettesítsük a változók minden előfordulását egyegy természetes számmal. Egy változó előfordulásához rendeljük hozzá a változót lekötő λ-absztrakciónak a változótól mért *hivatkozási távolságát*, ahol a hivatkozási távolság azon λ-absztrakciók száma, amelyek hatáskörében a változó benne van. Ha a változó egy λ-kifejezésben szabad változó, akkor egészítsük ki "virtuálisan" a λ-kifejezést egy legkülső, a változót kötő λ-absztrakcióval, és ezzel számítsuk ki a változó hivatkozási távolságát.

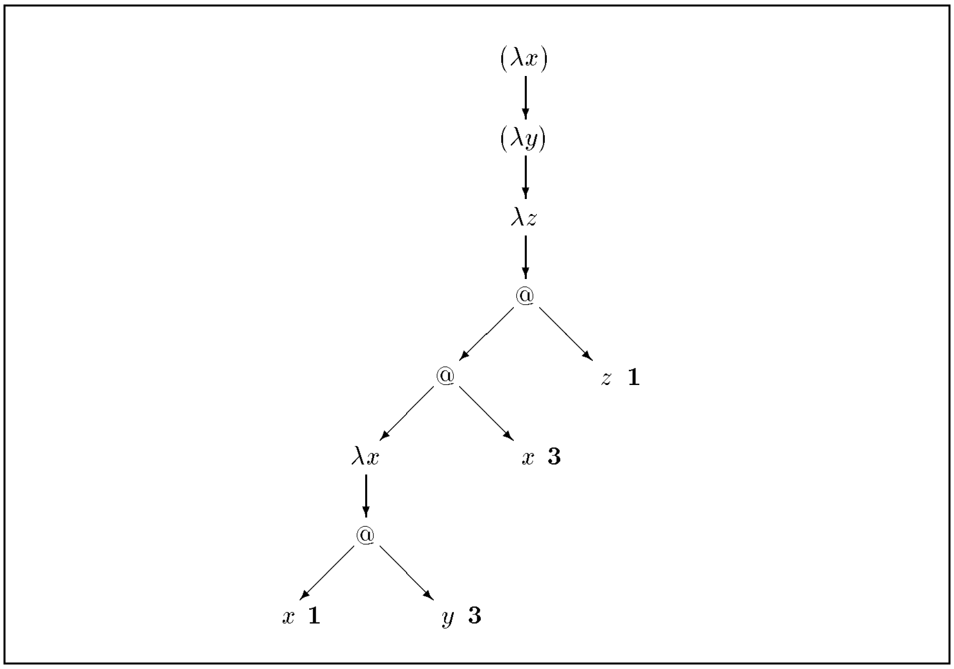
Az így kapott számokat *de Bruijn számoknak* nevezzük. A λ-kifejezések leírásából a λ-absztrakciók változóit is hagyjuk el, hiszen a de Bruijn számok meghatározása után a változók nevére már nincs szükség, de az egymásmelletti λ jeleket mindig írjuk ki. Ezt a formát a λ-kifejezés de Bruijn formájának nevezzük.

1.2.19. Példa. *(A de Bruijn számok meghatározása)*

$\lambda x.x$	$\lambda.1$,
$\lambda xy.xy$	$\lambda\lambda.2\ 1$,
$\lambda xy.xyz$	$\lambda\lambda.2\ 1\ 3$,
$\lambda z.(\lambda x.xy)xz$	$\lambda.(\lambda.1\ 3)\ 3\ 1$ *vagy* $\lambda.(\lambda.1\ 4)\ 2\ 1$,
$(\lambda y.(\lambda z.zy)y)((\lambda t.t)z)$	$(\lambda.(\lambda.1\ 2)\ 1)\ ((\lambda.1)\ 1)$. $\square$

A de Bruijn számok a λ-kifejezés gráfjából nagyon szemléletesen meghatározhatók. Egy vátozó de Bruijn száma azon λ gráfpontok darabszáma $+1$ lesz, amelyeken a változó pontjából a változót kötő λ-pontba vezető út keresztülmegy.

1.2.20. Példa. *(A de Bruijn számok meghatározása gráfból)*
Az 1.2. ábrán is látható $\lambda z.(\lambda x.xy)xz$ λ-kifejezés változóinak de Bruijn számait az 1.3. ábrán tüntetjük fel. A kifejezés $\lambda.(\lambda.1\,3)\;3\;1$ alakját rajzoljuk fel. □



1.3. ábra. A $\lambda z.(\lambda x.xy)xz$ λ-kifejezés változóinak de Bruijn számai

Ha E' és F' jelöli az E és F λ-kifejezések de Bruijn formáit, akkor a $(\lambda.E')\;F'$ β-redukció a

$$(\lambda.E')\;F' \;\rightarrow_\beta\; E'[1 := F']$$

formában írható fel, ahol $E'[1 := F']$ a következőket jelenti:

1. az E'-ben az E'-n kívüli kötésű változók de Bruijn számait eggyel csökkenteni kell, hiszen a β-redukció a $\lambda.E'$-ből egy λ-absztrakciót töröl.

2. az E'-ben meg kell keresni azokat a de Bruijn számokat, amelyeket F'-re kell kicserélni, és helyettesítéskor az F'-ben is módosítani kell egyes számokat, azokat, amelyek változóinak kötései az F'-n kívül vannak.

Ezek a műveletek a következő algoritmussal végezhetők el.

Először definiáljuk egy G' kifejezés n de Bruijn számának a G'-re számított $w_{G'}(n)$ *lokális súlyát*. A G' gráfjának minden p pontjához hozzárendelünk egy természetes számot. A G' gráf p_0 gyökérpontjából indulunk ki, és a fában felülről lefelé haladunk.

A p_0 gyökérpontban legyen $w_{G'}(p_0) = 1$. Ha a p ponthoz tartozó részfa G'_p kifejezésére $G'_p = G'_{p_1}\ G'_{p_2}$, akkor legyen $w_{G'}(p) = w_{G'}(p_1) = w_{G'}(p_2)$, és ha $G'_p = \lambda.G'_{p_1}$, akkor legyen $w_{G'}(p_1) = w_{G'}(p) + 1$.

A gráf levelein vannak a de Bruijn számok. Ha p_k levél az n de Bruijn számot tartalmazza, akkor a $w_{G'}(p_k)$ értéket nevezzük az n lokális súlyának, és ezt az értéket röviden $w_{G'}(n)$-nel jelöljük.

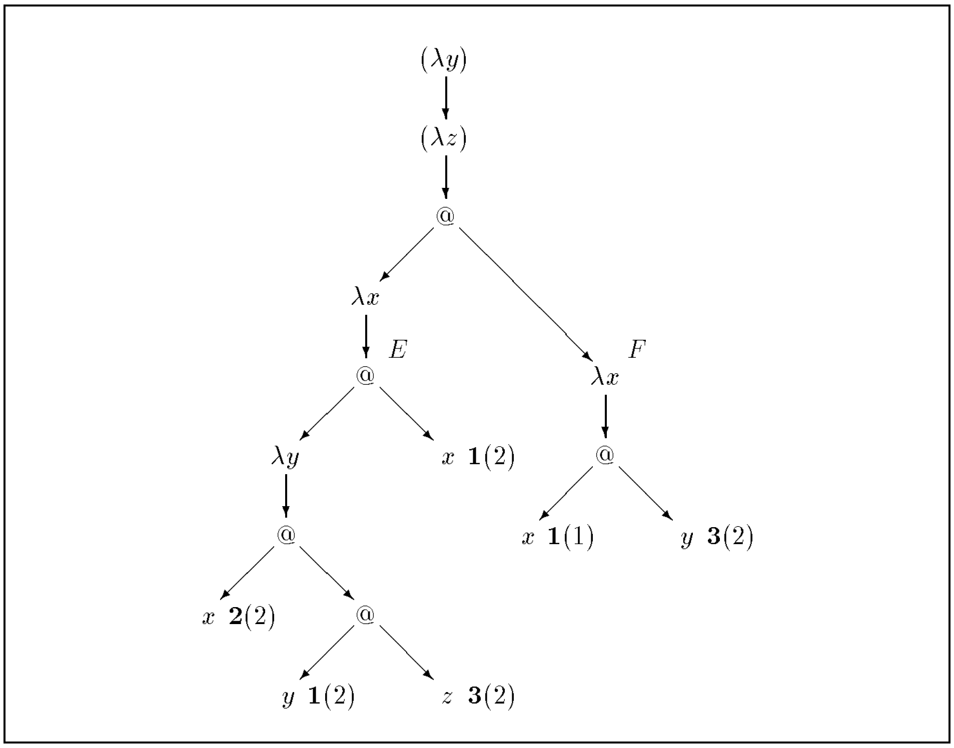
Ezt felhasználva, az $E'[1 := F']$ kifejezésben, ha $n > w_{E'}(n)$, akkor az n de Bruijn számnak megfelelő változó kötése E'-n kívül van, ha $n < w_{E'}(n)$, akkor az E' belsejében.

Ha $n = w_{E'}(n)$, akkor az n-nek megfelelő változót éppen a $\lambda.E'$ absztrakció köti, azaz az n-t az F' kifejezés módosított alakjával kell helyettesíteni. Tehát, ha n az E' egy de Bruijn száma, akkor $E'[1 := F']$-ben

$$n := \begin{cases} n-1, & \text{ha } n > w_{E'}(n), \\ n, & \text{ha } n < w_{E'}(n), \\ \mathcal{M}_n(F'), & \text{ha } n = w_{E'}(n), \end{cases}$$

ahol $\mathcal{M}_n(F')$ az F' kifejezés, módosított de Bruijn számokkal. Ha m az F' kifejezés egy de Bruijn száma, akkor az $\mathcal{M}_n(F')$ kifejezésben legyen

$$m := \begin{cases} m, & \text{ha } m < w_{F'}(m), \\ m+n-1, & \text{ha } m \geq w_{F'}(m). \end{cases}$$



1.4. ábra. A $(\lambda x.(\lambda y.x(yz))x)(\lambda x.xy)$ λ-kifejezés de Bruijn számai

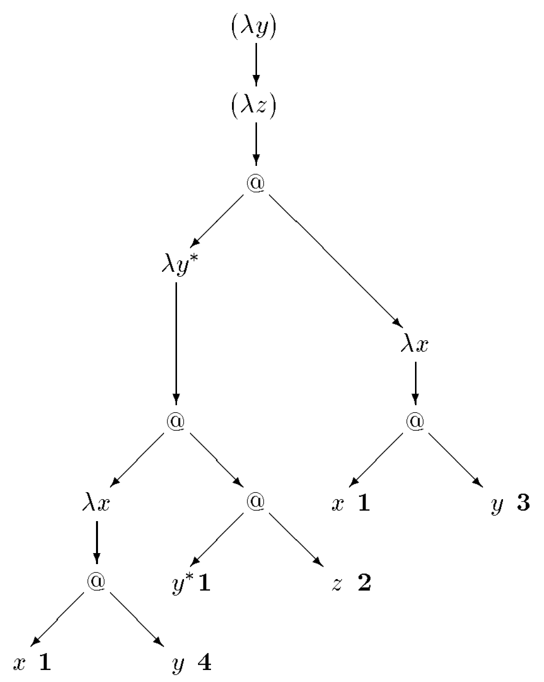
1.2.21. Példa. *(β-konverzió de Bruijn számokkal)*
Legyen

$$(\lambda x.E)F \equiv (\lambda x.(\lambda y.x(yz))x)(\lambda x.xy),$$

a kifejezés az 1.4. ábrán látható. A de Bruijn számok után zárójelben az E-re, illetve az F-re számított lokális súlyokat tüntettük fel.

A kifejezésre a β-konverzió nem hajtható végre, mert akkor az F y szabad változója kötötté válna. A de Bruijn alak β-konverziója után azonban nem lesz névkonfliktus (1.5. ábra):

$(\lambda.(\lambda.2\ (1\ 3))\ 1)\ (\lambda.1\ 3)\ \twoheadrightarrow_\beta$
$(\lambda.(\lambda.1\ 4)\ (1\ 2))\ (\lambda.1\ 3).$ $\square$



1.5. ábra. A $(\lambda y^*.(\lambda x.xy)\ (y^*z))\ (\lambda x.xy)$ λ-kifejezés de Bruijn számai

1.2.4. Egyenlőség

A továbbiakban konverzió alatt mindig β-konverziót értünk. Az α-konverziót szintaktikus átalakításnak tekintjük, az $E \leftrightarrow_\alpha F$ konverzióra az $E \equiv F$ jelölést használjuk, és ekkor az E és F λ-kifejezéseket nem tekintjük különböző λ-kifejezéseknek. Ha nem feltétlenül szükséges, a konverziók, redukciók és absztrakciók típusát nem jelöljük. A $\rightarrow^{+}$ és $\leftarrow^{+}$ jel redukciók és absztrakciók tranzitív lezárását, a $\rightarrow^{*}$ és $\leftarrow^{*}$ pedig a redukciók és absztrakciók reflexív és tranzitív lezárását jelöli.

Az 1.1.18. és az 1.2.13. definíciókkal λ-kifejezések szintaktikus azonosságát adtuk meg. Az előző pontokban láttuk, hogy egy λ-kifejezés konverziókkal egy másik λ-kifejezésbe alakítható át. Most két λ-kifejezés egyenlőségét definiáljuk. Az egyenlőséget az $=$ jellel jelöljük.

1.2.22. Definíció. Két λ-kifejezés egyenlősége:

Az E és F λ-kifejezésekre $E = F$, ha

- $E \equiv F$, *vagy*
- $E \leftrightarrow^{*} F$.

Ha $E = F$, akkor az E és az F egymásba *konvertálhatók*. Ha $E \equiv F$, akkor $E = F$, de a fordított irányú következtetés természetesen nem áll fenn.

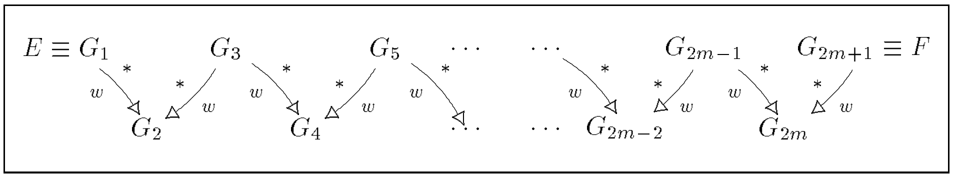
A definíció második pontja azt mondja ki, hogy ha $E = F$, akkor létezik olyan $F_1, F_2, \ldots, F_n$ sorozat ($n > 1$), amelyre

$E \equiv F_1$,
$F_i \leftrightarrow^{+} F_{i+1}$ vagy $F_i \equiv F_{i+1}$ $(1 \leq i \leq n-1)$,
$F_n \equiv F$.

Speciálisan (1.6. ábra), létezik olyan $G_1, G_2, \ldots, G_{2m}, G_{2m+1}$ sorozat ($m \geq 0$), amelyre

$E \equiv G_1$,
$G_{2j-1} \rightarrow^{*} G_{2j}$ vagy $G_{2j-1} \equiv G_{2j}$ $(1 \leq j \leq m)$,
$G_{2j} \leftarrow^{*} G_{2j+1}$ vagy $G_{2j} \equiv G_{2j+1}$ $(1 \leq j \leq m)$,
$G_{2m+1} \equiv F$.

A konverziós szabályok tulajdonságaiból azonnal látható, hogy az egyenlőség reláció *equivalencia reláció*, azaz az egyenlőség



1.6. ábra. Az $E = F$ egyenlőség.

- *reflexív*, $E = E$,
- *szimmetrikus*, ha $E = F$, akkor $F = E$,
- *tranzitív*, ha $E = F$ és $F = G$, akkor $E = G$.

Az egyenlőség relációval a Leibniz-szabály a következőképpen fogalmazható át:

1.2.23. (Leibniz-szabály:)

Ha $E_1 = F_1$, E_1 az E λ-kifejezés egy részkifejezése, és F az E-től csak abban különbözik, hogy benne az E_1 részkifejezés helyén az F_1 λ-kifejezés van, akkor $E = F$.

A Leibniz-szabály speciális esetei:

1.2.24. Következmény.

Ha $E = F$, akkor egy tetszőleges G λ-kifejezésre

1. $EG = FG$,
2. $GE = GF$,
3. $\lambda x.E = \lambda x.F$.

A ξ-szabály formája pedig:

1.2.25. Definíció. ξ-szabály:

A ξ-szabály a következő: ha $E = G$, akkor $\lambda x.E = \lambda x.G$.

Itt jegyezzük meg, hogy az $E = F$ reláció a benne szereplő konverziók elvégezhetőségét is jelenti. Ha ezt nem követelnénk meg, akkor meglepő eredményeket kapnánk:

1.2.26. Példa. *(Hibás egyenlőség, [Bar84] 25. oldal)*
Minden E és F λ-kifejezésre $E = F$.

„Bizonyítás." Legyen $G \equiv \lambda xy.yx$. Ekkor

$$GEF \equiv \underline{(\lambda xy.yx)EF} \to \underline{(\lambda y.yE)F} \to FE.$$

Ha $E \equiv y$ és $F \equiv x$, akkor ebből a levezetésből $Gyx = xy$. Ugyanakkor

$$Gyx \equiv \underline{(\lambda xy.yx)yx} \to \underline{(\lambda y.yy)x} \to xx,$$

azaz $xy = xx$. Az 1.2.24. következmény 2. pontját alkalmazzuk a $\lambda xy.y$ λ-kifejezésre,

$$\underline{(\lambda xy.y)xy} \to \underline{(\lambda y.y)y} \to y,$$

és

$$\underline{(\lambda xy.y)xx} \to \underline{(\lambda y.y)x} \to x,$$

azaz $x = y$. Ebből pedig már két tetszőleges λ-kifejezés egyenlősége is belátható. □

1.2.5. A λ-kalkulus axiómái

Miután áttekintettük a λ-kalkulus kifejezéseinek szintaktikáját és szemantikáját, megismertük a λ-kifejezések átalakítási, konverziós szabályait, most megadjuk a λ-kalkulus pontos definícióját.

1.2.27. Definíció. Az egyszerű típusnélküli λ-kalkulus:

Az egyszerű típusnélküli λ-kalkulus az egyszerű típusnélküli λ-kifejezések közötti olyan

$$E = F \quad (E, F \in \Lambda)$$

egyenlőségeket tartalmaz, amelyek a következő axiómák felhasználásával bizonyíthatók:

I.	$(\lambda x.E)F = E[x := F]$	*β-konverzió* (β)
II.i.	$E = E$	*reflexívitás* (ρ)
II.ii.	$E = F \Rightarrow F = E$	*szimmetria* (σ)
II.iii.	$E = F,\ F = G \ \Rightarrow\ E = G$	*tranzitívitás* (τ)
II.iv.	$E = F \ \Rightarrow\ EG = FG$	*Leibniz-szabály 1. következménye* (μ)
II.v.	$E = F \ \Rightarrow\ GE = GF$	*Leibniz-szabály 2. következménye* (ν)
II.vi.	$E = F \ \Rightarrow\ \lambda x.E = \lambda x.F$	ξ-szabály (ξ)

A λ-kalkulust gyakran λ-elméletnek, $\lambda\beta$-kalkulusnak, λK-kalkulusnak, $\lambda K\beta$-kalkulusnak is nevezik. A β elnevezés a β-konverzióból származik, a K pedig arra utal, hogy a K kifejezésnek jelentős szerepe van, mivel minden λ-kifejezés leírható K-val és a későbbiekben definiálandó λ_I-kifejezésekkel.

1.2.6. Az η-konverzió és a kiterjeszthetőség

Most bevezetünk egy új konverziós szabályt, az η-konverziót. Az η-konverzió azon alapul, hogy ha két függvény minden argumentumra megegyező eredményt ad, akkor a két függvény egyenlő. Az η-konverzió azt mondja ki, hogy ha egy λ-absztrakció törzse egy olyan függvényapplikáció, amelyben a paraméter a λ-absztrakció változója, akkor a λ-absztrakció az applikáció első tagjával helyettesíthető.

1.2.28. Definíció. Az η-konverzió:

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

A definíció első feltétele azt mondja ki, hogy az Ex egy függvényapplikáció, és a második feltétel szerint az E-ben nincs a λ-absztrakció x változójával kötött változó. Ha az első feltételt nem követelnénk meg, akkor az η-konverzió könnyen hibás eredményt adhatna, hiszen egy függvényt, azaz egy λ-absztrakciót η-konverzióval egy változóba, vagy akár egy applikációba lehetne átalakítani.

Az előzőekhez hasonlóan, az $E \leftrightarrow_\eta F$ azt jelenti, hogy az E egy részkifejezésére az η-konverziót alkalmazva az F-t kapjuk.

Az η-konverzió azt is bizonyítja, hogy bármely két tetszőleges λ-kifejezésre felírt applikáció függvényapplikációnak tekinthető. Az első kifejezésnek nem kell feltétlenül λ-absztrakciónak lennie, hiszen az η-konverzió alkalmazásával minden λ-kifejezés λ-absztrakcióvá alakítható át. Az első λ-kifejezésre alkalmazott η-konverzió után az applikáció biztosan függvényapplikáció lesz:

$EF \leftrightarrow_\eta (\lambda x.Ex)F.$

1.2.29. Példa. *(η-konverziók)*

$\lambda x.(\lambda x.xy)x \leftrightarrow_\eta \lambda x.xy,$

$\lambda xx.xyyzx \leftrightarrow_\eta \lambda x.xyyz.$ □

1.2.30. Definíció. Az egyszerű típusnélküli $\lambda\eta$-kalkulus:

A $\lambda\eta$-kalkulust úgy kapjuk meg, hogy a λ-kalkulus axiómáit kiegészítjük a

II.vii. $\lambda x.Ex = E$ *η-konverzió (η)*

axiómával.

Ha $Ex \rightarrow_\beta G$, $Fx \rightarrow_\beta G$, azaz két azonos x aktuális paraméterű λ-kifejezés ugyanarra a λ-kifejezésre konvertálható, akkor az η-konverzió felhasználásával belátható, hogy a két λ-kifejezés egymásba is transzformálható, feltéve, hogy az x vátozó nem szabad változó sem E-ben, sem F-ben:

$E \leftrightarrow_\eta \lambda x.Ex \rightarrow_\beta \lambda x.G,$

$F \leftrightarrow_\eta \lambda x.Fx \rightarrow_\beta \lambda x.G,$

azaz

$E \leftrightarrow^+_{\beta,\eta} F.$

Ezt a tulajdonságot kiterjeszthetőségnek nevezzük.

1.2.31. Definíció. Kiterjeszthetőség:

Ha $Ex = Fx$, és $x \notin FV(E)$, $x \notin FV(F)$, akkor $E = F$.

A kiterjeszthetőség, az η-konverzióhoz hasonlóan, tehát azon alapul, hogy ha két függvény értéke minden argumentumra azonos, akkor a két függvény is megegyezik.

1.2.32. Definíció. Az egyszerű típusnélküli λ+*ext*-kalkulus:

A λ+ext-kalkulust úgy kapjuk meg, hogy a λ-kalkulus axiómáit kiegészítjük a

II.vii. $Ex = Fx \;\Rightarrow\; E = F$ *kiterjeszthetőség (ext)*

axiómával.

A ξ-szabály és az η-konverzió alkalmazásával, azaz a $\lambda\eta$-kalkulusban bizonyítani tudjuk a kiterjeszthetőségnek nevezett tulajdonságot: ha az $E_1x = E_2x$ -re a ξ -szabályt alkalmazzuk, akkor

$\lambda x.E_1x = \lambda x.E_2x,$

és az η-konverzióval

$E_1 = E_2,$

tehát az η-konverzióval a kiterjeszthetőség bizonyítható.

Az állítás a fordított irányban is igaz, a kiterjeszthetőségből az η-konverzió levezethető, azaz a λ+*ext*-kalkulusban az η-konverzió bizonyítható: ha $x \notin FV(E)$, a

$(\lambda x.Ex)x = Ex$

egyenlőségre a kiterjeszthetőséget alkalmazva

$\lambda x.Ex = E,$

azaz éppen az η-konverziót kapjuk eredményül.

Tehát megállapíthatjuk, hogy az η-konverzió helyett a kiterjeszthetőséget is használhatjuk a λ-kalkulus konverziós szabályaként. Ez az állítás Curry-től származik:

1.2.33. Tétel. (Curry-tétel)

A $\lambda\eta$-kalkulus és a λ+ext-kalkulus ekvivalensek.

Mivel a ξ-szabály jelentős szerepet játszik mindkét levezetésben, a ξ-szabályt a *gyenge kiterjeszthetőség szabályának* is szokás nevezni.

1.3. A λ-kifejezés normál formája

Az előző pontokban már láttuk, hogy egy λ-kifejezés konverziók, speciálisan redukciók sorozatával egy másik λ-kifejezésbe alakítható át. A redukciósorozat minden lépésében egy *redukálható kifejezést*, *redexet* redukálunk. Ha az E_1 λ-kifejezést redukciók sorozatával az E_2 λ-kifejezésre alakítjuk át, azaz $E_1 \twoheadrightarrow^* E_2$, akkor azt mondjuk, hogy E_2 az E_1 *redukáltja.*

1.3.1. Definíció. Normál forma:

Ha egy λ-kifejezésben nincs redukálható kifejezés, akkor a λ-kifejezés normál formában van.

Ha egy λ-kifejezés redukciók sorozatával normál formára hozható, akkor azt mondjuk, hogy a λ-kifejezésnek *van normál formája.* Megjegyezzük, hogy mivel az α-konverziót nem tekintjük redukciónak, a normál formából α-konverzióval kapott λ-kifejezést nem tekintjük az eredetitől különböző normál formának.

Ha a λ-kalkulusban egy λ-kifejezés normál formában van, akkor nincs $(\lambda x.E)F$ alakú részkifejezése, a $\lambda\eta$-kalkulusban pedig nincs sem $(\lambda x.E)F$, sem $\lambda x.Gx$, $(x \notin FV(G))$ alakú részkifejezése. A két normál forma közötti kapcsolatot Curry bizonyította be:

1.3.2. Tétel. (Normál formák tétele, Curry-tétel [1972.])

Az $E \in \Lambda$ λ-kifejezésnek a λ-kalkulusban akkor és csak akkor van normál formája, ha a $\lambda\eta$-kalkulusban is van normál formája.

A fenti tételből következik, hogy a normál forma tulajdonságainak vizsgálatakor a továbbiakban elegendő csak a λ-kalkulussal foglalkoznunk.

1.3.3. Példa. *(A következő λ-kifejezések normál formában vannak)*

x,

xy,

$\lambda x.y$,

$\lambda x.(\lambda y.z)$. □

1.3.4. Példa. *(A következő λ-kifejezés nincs normál formában, de normál formára hozható)*

$\underline{(\lambda x.x)(\lambda x.x)} \twoheadrightarrow_\beta \lambda x.x$ □

Ha egy λ-kifejezés nincs normál formában, akkor nem feltétlenül hozható normál formára:

1.3.5. Példa. *(A következő λ-kifejezés nincs normál formában és a λ-kifejezésnek nincs normál formája)*

$\underline{(\lambda x.xx)(\lambda x.xx)} \twoheadrightarrow_\beta$

$\underline{(\lambda x.xx)(\lambda x.xx)} \twoheadrightarrow_\beta$

$\underline{(\lambda x.xx)(\lambda x.xx)} \twoheadrightarrow_\beta \ \ldots$

A redukciósorozat nem terminál, hiszen a redukcióval mindig az eredeti kifejezést kapjuk vissza. □

A fenti példában szereplő λ-kifejezés a β-redukcióval önmagát reprodukálja, ezt a λ-kifejezést a továbbiakban Ω-val jelöljük:

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

Church az olyan λ-kifejezéseket, amelyeknek van normál formájuk, „*jelentéssel bíró*”, vagy egyszerűen csak *jelentős* λ-kifejezéseknek nevezte. Ezeknek a λ-kifejezéseknek viszont fennáll az a furcsa tulajdonságuk, hogy lehetnek olyan részkifejezéseik, amelyeknek „nincs jelentésük”:

1.3.6. Példa. *(„Jelentéssel bíró” λ-kifejezés)*
A $\mathsf{KI}\Omega$ λ-kifejezésnek van „jelentése”:

$\mathsf{KI}\Omega \twoheadrightarrow_\beta \mathsf{I}$, de az Ω részkifejezésnek nincs normál formája, azaz „nincs jelentése". □

Sőt mi több, egy még furcsább tulajdonság: a λ-kalkulusban az is előfordul, hogy ha egy „jelentéssel nem bíró" λ-kifejezésre, mint függvényre egy E λ-kifejezést applikálunk, az, hogy az eredménynek van-e normál formája, azaz van-e „jelentése", az E-től függ. Ha egy „jelentéssel nem bíró" λ-kifejezésre egy jelentős λ-kifejezést alkalmazunk, akkor eredményül „jelentéssel bíró" függvényt is kaphatunk. Egy ilyen „jelentéssel nem bíró" függvény például az Y fixpont-kombinátor (1.5.1. pont).

Church a λ-kalkulusban a „jelentéssel bíró" λ-kifejezéseknek központi szepet adott, és azoknak a λ-kifejezéseknek, amelyeknek nincs normál formájuk, a „nincs értelmezve" függvényfogalmat feleltette meg. Ebből azonban, mint majd az 1.6.3. pontban látni fogjuk, több komoly, a λ-kalkulus inkonzisztenciáját okozó probléma származott. Church végül is ezt a konfliktust úgy oldotta meg, hogy definiálta a λ_I-kalkulust, amelyben a fenti tulajdonságok már nem fordulhatnak elő.

1.3.1. Redukálási stratégiák

Ha egy λ-kifejezésnek több redexe is van, akkor a redexek különböző redukálási sorrendjei különböző redukálási sorozatokat határoznak meg.

Több redex esetén lehetséges, hogy a különböző redukálási sorozatok ugyanahhoz a normál formához vezetnek, de az is lehetséges, hogy az egyik redukálási sorozattal nem kapjuk meg a λ-kifejezés normál formáját, egy másik sorozattal viszont a normál formához jutunk.

1.3.7. Példa. *(Különböző redukálási sorrenddel ugyanazt a normál formát kapjuk)*
Először redukáljunk az u paraméterrel:

$\underline{(\lambda x.((\lambda y.xy)((\lambda z.z)v)))u} \twoheadrightarrow_\beta$
$\underline{(\lambda y.uy)\ ((\lambda z.z)v)} \twoheadrightarrow_\beta$
$u(\underline{(\lambda z.z)v}) \twoheadrightarrow_\beta uv.$

Most először redukáljunk a v paraméterrel:

$(\lambda x.((\lambda y.xy)(\underline{(\lambda z.z)v})))u \ \twoheadrightarrow_\beta$
$(\lambda x.(\underline{(\lambda y.xy)v})u \ \twoheadrightarrow_\beta$
$\underline{(\lambda x.(xv))u} \ \twoheadrightarrow_\beta \ uv.$ □

1.3.8. Példa. *(Az első applikáció végrehajtásával normál formát kapunk, a második, belső applikáció végrehajtásakor az 1.3.5. példában is szereplő végtelen ciklust)*

$(\lambda y.z)\Omega \equiv$
$\underline{(\lambda y.z)((\lambda x.xx)\ (\lambda x.xx))} \ \twoheadrightarrow_\beta \ z,$
és
$(\lambda y.z)\Omega \equiv$
$(\lambda y.z)(\underline{(\lambda x.xx)\ (\lambda x.xx)}) \ \twoheadrightarrow_\beta$
$(\lambda y.z)(\underline{(\lambda x.xx)\ (\lambda x.xx)}) \ \twoheadrightarrow_\beta$
$(\lambda y.z)\Omega \ \twoheadrightarrow_\beta \ (\lambda y.z)\Omega \ \twoheadrightarrow_\beta \ \ldots$ □

A λ-kifejezés normál formája a λ-kifejezés „értékének" is tekinthető, mivel a λ-kifejezés további redukálása már nem lehetséges. A következő tétel és a tétel következménye ennek az értéknek ez egyértelműségét mondja ki.

1.3.9. Tétel. (Az I. Church–Rosser-tétel)

Ha $E_1 = E_2$, akkor létezik egy olyan F λ-kifejezés, amelyre $E_1 \twoheadrightarrow^* F$ és $E_2 \twoheadrightarrow^* F$.

Bizonyítás. A tételt nem bizonyítjuk, a bizonyítás [Bar84] és [Hin86]-ban megtalálható. A bizonyítás alapja a következő állítás:

Ha $E \leftrightarrow^* E_1$ és $E \leftrightarrow^* E_2$, akkor létezik egy olyan F λ-kifejezés, amelyre $E_1 \twoheadrightarrow^* F$, és $E_2 \twoheadrightarrow^* F$.

Ez az állítás az 1.7. ábrán látható, a gráf alapján az I. Church–Rosser-tételben kimondott tulajdonságot gyakran *rombusz-tulajdonságnak* is nevezik. Az $E_1 = E_2$ egyenlőségben szereplő konverziókra (1.6. ábra) a rombusz-tulajdonságot alkalmazva a tétel állítása bizonyítható (1.8. ábra). □

$$\begin{tikzcd} & E \arrow[dl, shift left] \arrow[dr, shift left] & \\ E_1 \arrow[ur, shift left] \arrow[dr] & & E_2 \arrow[ul, shift left] \arrow[dl] \\ & F & \end{tikzcd}$$

1.7. ábra. A rombusz-tulajdonság

Az I. Church–Rosser-tétel tehát azt mondja ki, hogy két egymásba transzformálható λ-kifejezéshez mindig létezik, esetleg az egyikkel meg is egyező λ-kifejezés, amelybe mindkét λ-kifejezés átkonvertálható. A tételből két egyszerűen bizonyítható állítás következik:

1.3.10. Következmény.

Ha $E_1 = E_2$, és E_2 normál formában van, akkor $E_1 \rightarrow\!\!\!\triangleright^* E_2$.

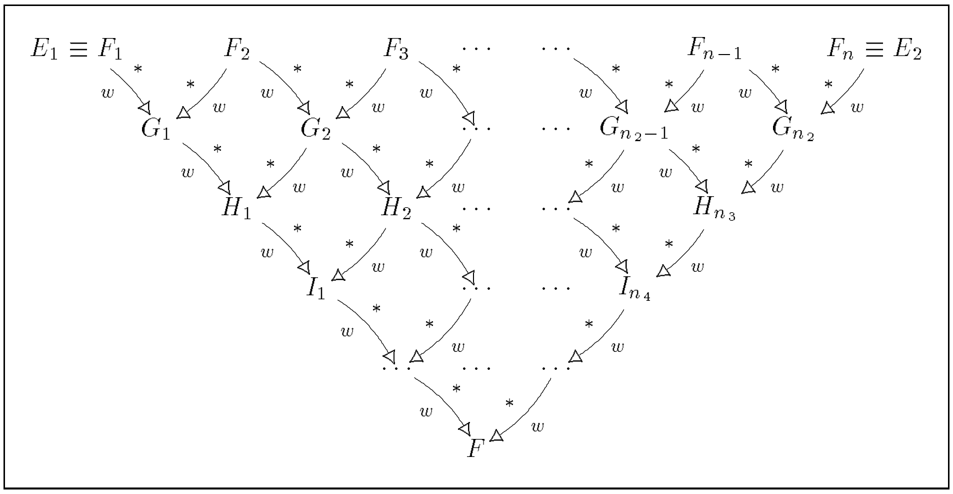
A következmény tehát azt mondja ki, hogy ha egy λ-kifejezésnek létezik normál formája, akkor a normál forma a λ-kifejezésből redukciók sorozatával előállítható.

1.3.11. Következmény.

Minden λ-kifejezésnek legfeljebb egy normál formája van.

Bizonyítás. Tegyük fel, hogy az E λ-kifejezésnek az F_1 és F_2 két különböző normál formája. Ekkor az I. Church–Rosser-tétel alapján létezik olyan F λ-kifejezés, amelyre $F_1 \rightarrow\!\!\!\triangleright^* F$ és $F_2 \rightarrow\!\!\!\triangleright^* F$. Az F_1 és az F_2 normál formák, azaz tovább nem redukálhatók, így csak az $F_1 = F_2 = F$ lehetséges. $\square$

A fenti két következményből azonnal következik a következő tulajdonság:



1.8. ábra. Az I. Church–Rosser-tétel

1.3.12. Következmény.

Ha E és F mindegyike normál forma, és $E \not\equiv F$, akkor $E \neq F$.

A fentiek szerint tehát, ha eljutunk a normál formához, akkor már közömbös, hogy milyen redukciók eredményeként kaptuk azt meg, hiszen minden más normál formára hozó redukció ugyanezt az eredményt adja. Ez a λ-kalkulus *konzisztens* tulajdonságát mutatja, azaz minden olyan sorozat, amelyik egy λ-kifejezés normál formáját állítja elő, ugyanahhoz a λ-kifejezéshez vezet. Ugyanakkor a fentiekből az is következik, hogy a λ-kalkulus *konfluens*, azaz egy λ-kifejezéshez nincs két olyan redukciós sorozat, amely különböző normál formákat állítana elő.

A *redukálási stratégia* meghatározza, hogy több redex esetén melyik redexet kell redukálni, azaz a redukálási stratégia egy *redukálási sorrendet* ad meg. Ha egy λ-kifejezésnek létezik normál formája, és egy redukálási stratégia olyan redukálási sorrendet határoz meg, amellyel a λ-kifejezés normál formáját kapjuk meg, akkor a redukálási stratégiát *normalizáló redukálási stratégiának* nevezzük.

Egy λ-kifejezés legkülső redexe az a redukálható kifejezése, amelyik nincs más redexének a belsejében, és a legbelső redexe az a redukálható kifejezése, amelyik belsejében nem tartalmaz redukálható kifejezést. Négy redukálási stratégiát szokás megkülönböztetni, ezek egy λ-kifejezésnek mindig a

- legbaloldalibb legkülső,
- legbaloldalibb legbelső,
- legjobboldalibb legkülső,
- legjobboldalibb legbelső

redexét redukálják. Egy λ-kifejezésnek ezek a redexei legszemléletesebben a λ-kifejezés gráfjából határozhatók meg, ezt a következő példában mutatjuk meg.

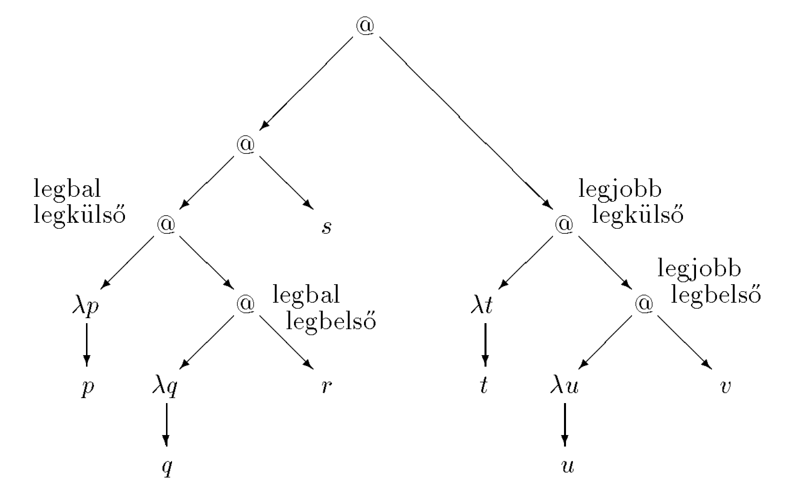
1.3.13. Példa. *(Redukálási stratégiák redexei)*
A λ-kifejezés gráfjából a redexek könnyen meghatározhatók (1.9. ábra). □

Normál sorrendű redukálási stratégia

Azt a redukálási stratégiát, amelyik a legbaloldalibb legkülső redexet redukálja, *normál sorrendű redukálási stratégiának* nevezzük, és azt mondjuk, hogy alkalmazásával *normál sorrendű* redukálásokat hajtunk végre.

A normál sorrendű redukálás függvényapplikációkra először a függvény λ-kifejezését redukálja addig, amíg benne redexet talál, és csak ezután foglalkozik, amikor már feltétlenül szükséges, az aktuális paraméter értékének meghatározásával. Ezt a módszert *lusta paraméterkiértékelésnek* is nevezzük. Megjegyezzük, hogy ez a paraméterkiértékelés nem véges adatstruktúrák konstrukcióját és kezelését is lehetővé teszi. Ez a paraméterátadási módszer az imperatív nyelvek *névszerinti paraméterátadásának* felel meg.

A normál sorrendű redukálás jelentőségét a következő tétel adja meg, a tételt a *normalizálás tételének* is nevezzük.



1.9. ábra. A $((\lambda p.p)((\lambda q.q)r))s((\lambda t.t)((\lambda u.u)v))$ λ-kifejezés gráfja

1.3.14. Tétel. (A II. Church–Rosser-tétel)

A normál sorrendű redukálási stratégia mindig normalizáló redukálási stratégia.

A tétel szerint, ha egy λ-kifejezésnek van normál formája, akkor a normál forma normál sorrendű redukálási stratégiával előállítható. Megjegyezzük azonban, hogy a normál forma meghatározása nem feltétlenül a legkevesebb lépésben történik. Megadhatók olyan stratégiák is, amelyek a legkevesebb lépésszámban állítják elő a normál formát, de az optimális lépés meghatározása gyakran több munkát igényel, mint amit a normál sorrendű redukálás több lépésszáma okoz. A normál sorrendű redukálás gyakran meglepően alacsony hatásfokú:

1.3.15. Példa. *(A normál sorrendű redukálás hatékonysága)*
Legyen twotimes $\equiv \lambda x.\mathsf{add}\ xx$, ekkor az E négyszeresét a következő módon számíthatjuk ki:

$\mathsf{twotimes}(\mathsf{twotimes}\ E) \equiv$

$\underline{(\lambda x.\mathsf{add}\ xx)(\mathsf{twotimes}\ E)} \ \twoheadrightarrow_\beta$

$\mathsf{add}(\mathsf{twotimes}\ E)(\mathsf{twotimes}\ E) \ \twoheadrightarrow_\beta^+$

$\mathsf{add}(\mathsf{add}\ EE)(\mathsf{add}\ EE)$,

azaz ugyanazt az $\mathsf{add}\ EE$ műveletet kétszer is el kell végezni. □

Érdekességképpen megemlítjük, hogy az imperatív nyelvek *if-then-else* utasítása a lusta paraméterkiértékelés tipikus példája, hiszen ha a feltétel igaz, az *else*, ellenkező esetben pedig a *then* ág programja nem értékelődik ki, azaz nem hajtódik végre.

Ha egy λ-kifejezésnek nincs normál formája, akkor a λ-kifejezés redukálása normál sorrendű redukálással véges lépésben nem fejeződik be.

Applikatív sorrendű redukálási stratégia

Azt a redukálási stratégiát, amelyik a legbaloldalibb legbelső redexet redukálja, *applikatív sorrendű redukálási stratégiának* nevezzük, és azt mondjuk, hogy ilyenkor a redukálásokat *applikatív sorrendben* hajtjuk végre. Az elnevezés onnan származik, hogy ez a redukálási stratégia függvényapplikációkban először mindig a függvény argumentumát redukálja, azaz először az aktuális paramétert értékeli ki, és csak az argumentum meghatározása után foglalkozik a függvény λ-kifejezésének redukálásával. Ezt a paraméterátadási módszert, szemben a lusta kiértékeléssel, *mohó paraméterkiértékelésnek* nevezzük, és ez a paraméterátadási módszer az imperatív nyelvek *érték szerinti paraméterátadásának* felel meg.

Ha egy λ-kifejezésnek van normál formája, akkor a normál formát az applikatív sorrendű redukálási stratégia nem feltétlenül határozza meg.

1.3.16. Példa. *(Applikatív sorrendű redukálás)*
Az 1.3.7. és az 1.3.8. példákban az első redukálás normál sorrendű, a második redukálás applikatív sorrendű redukálás volt. Mindkét λ-kifejezésnek volt normál formája, de az 1.3.8. példa applikatív sorrendű redukálása a normál formát nem határozta meg. □

1.4. Konstansok és konstans függvények

Az egyszerű típusnélküli λ-kalkulusban nincsenek konstansok, és nincsenek konstansokon értelmezett függvények, de az egyszerű típusnélküli λ-kalkulus egyes λ-kifejezéseit konstansoknak és a konstansokon értelmezett függvényeknek tekinthetjük, és ezekre a λ-kifejezésekre a szokásos aritmetikai, logikai tulajdonságok is teljesülnek. Ez a λ-kalkulus leíró erejét mutatja, és azt jelenti, hogy a funkcionális programok a λ-kalkulus kifejezéseire konvertálhatók. A funkcionális programok logikai és aritmetikai konstansainak és függvényeinek konvertálása tehát nem okoz problémát, a λ-kalkulust ehhez nem kell bővíteni.

1.4.1. A logikai konstansok és műveletek

Reprezentálják a logikai konstansokat a következő λ-kifejezések:

$\mathsf{true} \equiv \lambda xy.x,$

$\mathsf{false} \equiv \lambda xy.y.$

Látható, hogy a logikai konstansok nem értékek, hanem λ-absztrakciók, a **true** az első, a **false** a második aktuális paramétert adja eredményül. Ez éppen megfelel a szokásos *if-then-else* utasításnak: ha a logikai konstans λ-absztrakciója az *if* utasítás feltétele, akkor a *then*-ág kifejezése a λ-absztrakció első, az *else*-ág kifejezése a λ-absztrakció második aktuális paraméterének tekinthető. Ezért az if nevű *if-then-else* λ-absztrakciót az

$\mathsf{if} \equiv \lambda pqr.pqr$

alakban definiálhatjuk.

1.4.1. Példa. *(Az* if true EF *applikáció az* E *kifejezésre, az* if false EF *pedig az* F*-re redukálható)*

$\mathsf{if}\ \mathsf{true}\ EF \equiv$

$\underline{(\lambda pqr.pqr)\mathsf{true}}\ EF \ \twoheadrightarrow_\beta$

$\underline{(\lambda qr.\mathsf{true}\ qr)EF} \ \twoheadrightarrow_\beta$

$\underline{(\lambda r.\mathsf{true}\ Er)F} \twoheadrightarrow_\beta$
$\mathsf{true}\ EF \equiv$
$\underline{(\lambda xy.x)EF} \twoheadrightarrow_\beta$
$\underline{(\lambda y.E)F} \twoheadrightarrow_\beta E,$

$\mathsf{if\ false}\ EF \equiv$
$\underline{(\lambda pqr.pqr)\mathsf{false}}\ EF \twoheadrightarrow_\beta$
$\underline{(\lambda qr.\mathsf{false}\ qr)E}F \twoheadrightarrow_\beta$
$\underline{(\lambda r.\mathsf{false}\ Er)F} \twoheadrightarrow_\beta$
$\mathsf{false}\ EF \equiv$
$\underline{(\lambda xy.y)EF} \twoheadrightarrow_\beta$
$\underline{(\lambda y.y)F} \twoheadrightarrow_\beta F.$ □

Mivel már ismerjük az if λ-kifejezést, a konstansokon értelmezett logikai függvények értékének meghatározására használhatjuk az imperatív nyelvekből jól ismert *optimalizált kiértékelés* módszerét. Az optimalizált logikai kiértékelés és az if definíciója szerint

$\mathsf{and}\ EF \approx \mathsf{if}\ EF\ \mathsf{false} \equiv (\lambda pqr.pqr)EF\ \mathsf{false} \twoheadrightarrow_\beta^+ EF\ \mathsf{false},$
$\mathsf{or}\ EF \approx \mathsf{if}\ E\ \mathsf{true}\ F \equiv (\lambda pqr.pqr)E\ \mathsf{true}\ F \twoheadrightarrow_\beta^+ E\ \mathsf{true}\ F,$

és hasonlóan

$\mathsf{not}\ E \approx \mathsf{if}\ E\ \mathsf{false\ true} \equiv (\lambda pqr.pqr)E\ \mathsf{false\ true} \twoheadrightarrow_\beta^+ E\ \mathsf{false\ true}.$

Ezekből pedig a logikai függvények λ-absztrakciói:

$\mathsf{and} \equiv \lambda xy.xy\ \mathsf{false},$
$\mathsf{or} \equiv \lambda xy.x\ \mathsf{true}\ y,$
$\mathsf{not} \equiv \lambda x.x\ \mathsf{false\ true}.$

1.4.2. Példa. *(Az* and false true *kifejezés)*
$\mathsf{and\ false\ true} \equiv$
$\underline{(\lambda xy.xy\ \mathsf{false})\mathsf{false}}\ \mathsf{true} \twoheadrightarrow_\beta$

$\underline{(\lambda y.\mathsf{false}\ y\ \mathsf{false})\mathsf{true}} \rightarrow_\beta$
$\mathsf{false}\ \mathsf{true}\ \mathsf{false} \equiv$
$\underline{(\lambda xy.y)\mathsf{true}}\ \mathsf{false} \rightarrow_\beta$
$\underline{(\lambda y.y)\mathsf{false}} \rightarrow_\beta \mathsf{false}.$ □

1.4.3. Példa. *(A* not false *kifejezés)*

$\mathsf{not}\ \mathsf{false} \equiv \underline{(\lambda x.x\ \mathsf{false}\ \mathsf{true})\mathsf{false}} \rightarrow_\beta$
$\mathsf{false}\ \mathsf{false}\ \mathsf{true} \equiv$
$\underline{(\lambda xy.y)\mathsf{false}}\ \mathsf{true} \rightarrow_\beta$
$\underline{(\lambda y.y)\mathsf{true}} \rightarrow_\beta \mathsf{true}.$ □

1.4.2. Pár, n-es és műveleteik

A *rendezett pár* kombinátora legyen a pair λ-kifejezés, és válassza ki a rendezett pár első elemét a first, a második elemét a second függvény. Ezekre teljesülnie kell a következőknek:

$\mathsf{first}(\mathsf{pair}\ uv) \rightarrow_\beta^+ u,$
$\mathsf{second}(\mathsf{pair}\ uv) \rightarrow_\beta^+ v,$

azaz a pair függvényt aktuális paraméterként alkalmazva a first és second projekciókra, az applikációnak éppen a megfelelő komponenst kell eredményül adnia. Ezek a függvények a következőképpen írhatók le λ-absztrakcióval:

$\mathsf{pair} \equiv \lambda xyz.zxy,$
$\mathsf{first} \equiv \lambda x.x\ \mathsf{true} \equiv \lambda x.x(\lambda yz.y),$
$\mathsf{second} \equiv \lambda x.x\ \mathsf{false} \equiv \lambda x.x(\lambda yz.z).$

Könnyen belátható, hogy az így definiált függvények kielégítik a fenti tulajdonságokat.

A pair EF kifejezésre

$\mathsf{pair}\ EF \rightarrow^+ \lambda z\,.\,zEF,$

ezt a kifejezést gyakran $[E, F]$-fel, vagy $\langle E, F\rangle$-fel is jelölik.

1.4.4. Példa. *(A* $\mathsf{first}(\mathsf{pair}\ uv) \twoheadrightarrow^{+} u$ *redukció)*

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

$\underline{(\lambda x.x(\lambda yz.y))(\mathsf{pair}\ uv)} \twoheadrightarrow_\beta$

$\mathsf{pair}\ uv(\lambda yz.y) \equiv$

$\underline{(\lambda xyz.zxy)}uv(\lambda yz.y) \twoheadrightarrow_\beta$

$\underline{(\lambda yz.zuy)v}(\lambda yz.y) \twoheadrightarrow_\beta$

$\underline{(\lambda z.zuv)(\lambda yz.y)} \twoheadrightarrow_\beta$

$\underline{(\lambda yz.y)uv} \twoheadrightarrow_\beta$

$\underline{(\lambda z.u)v} \twoheadrightarrow_\beta u.$ □

1.4.5. Példa. *(*$\mathsf{second}(\mathsf{pair}\ uv) \twoheadrightarrow^{+} v$*)*

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

$\underline{(\lambda x.x(\lambda yz.z))(\mathsf{pair}\ uv)} \twoheadrightarrow_\beta$

$\mathsf{pair}\ uv\ (\lambda yz.z) \equiv$

$\underline{(\lambda xyz.zxy)}uv(\lambda yz.z) \twoheadrightarrow_\beta$

$\underline{(\lambda yz.zuy)v}(\lambda yz.z) \twoheadrightarrow_\beta$

$\underline{(\lambda z.zuv)(\lambda yz.z)} \twoheadrightarrow_\beta$

$\underline{(\lambda yz.z)uv} \twoheadrightarrow_\beta$

$\underline{(\lambda z.z)v} \twoheadrightarrow_\beta v.$ □

Az n-eseket ($n \geq 2$) a pair $n-1$-szeri alkalmazásával kaphatjuk meg. Ha $\vec{x} \equiv x_1 x_2 \ldots x_n$, akkor az $[E_1, E_2, \ldots, E_n]$-nel jelölt n-es konstruktora

$$\mathsf{n\text{-}es} \equiv \lambda\vec{x}.\mathsf{pair}\ x_1(\mathsf{pair}\ x_2(\ldots(\mathsf{pair}\ x_{n-1}x_n)\ldots)),$$

azaz

$$[E_1, E_2, \ldots, E_n] \equiv [E_1, [E_2, \ldots, [E_{n-1}, E_n]\ldots]] \equiv$$
$$\lambda z.zE_1(\lambda z.zE_2(\ldots(\lambda z.zE_{n-1}E_n)\ldots)).$$

Egy n-es i-edik elemét a

$$\mathsf{select_i} \equiv \begin{cases} \lambda x.\mathsf{first}(\mathsf{second}^{i-1}(x)), & \text{ha } 1 \le i < n, \\ \lambda x.\mathsf{second}^{n-1}(x), & \text{ha } i = n \end{cases}$$

λ-absztrakcióval határozhatjuk meg, ahol

$$f^n(x) \equiv \begin{cases} f\ (f^{n-1}(x)), & \text{ha } n \ge 1, \\ x, & \text{ha } n = 0. \end{cases}$$

Az i-edik elemet kiválasztó szelektort megadhatjuk a következő λ-kifejezéssel is:

$$\mathsf{select_i} \equiv \begin{cases} \lambda x.x\ \mathsf{false}^{\sim(i-1)}\mathsf{true}, & \text{ha } 1 \le i < n, \\ \lambda x.x\ \mathsf{false}^{\sim n}, & \text{ha } i = n, \end{cases}$$

ahol

$$x f^{\sim n} \equiv \begin{cases} (x f^{\sim(n-1)})f, & \text{ha } n \ge 1, \\ x, & \text{ha } n = 0. \end{cases}$$

Az első sor az applikációk asszociatívitása miatt $xf^{\sim(n-1)}f$ alakban is írható.

1.4.6. Példa. *(Az $[E_1, E_2, E_3, E_4]$ négyes harmadik komponense)*
A négyes λ-kifejezése $\mathsf{pair}\ E_1(\mathsf{pair}\ E_2(\mathsf{pair}\ E_3E_4))$, és a harmadik komponenst kiválasztó szelektor a másodikként megadott λ-kifejezéssel

$\lambda x.x\ \mathsf{false}^{\sim 2}\mathsf{true}$.

Ekkor

$$\underline{(\lambda x.x\ \mathsf{false}^{\sim 2}\mathsf{true})(\mathsf{pair}\ E_1(\mathsf{pair}\ E_2(\mathsf{pair}\ E_3E_4)))} \twoheadrightarrow_\beta$$
$$(\mathsf{pair}\ E_1(\mathsf{pair}\ E_2(\mathsf{pair}\ E_3E_4)))\mathsf{false}^{\sim 2}\mathsf{true} \equiv$$
$$(\mathsf{pair}\ E_1(\mathsf{pair}\ E_2(\mathsf{pair}\ E_3E_4)))\mathsf{false}\ \mathsf{false}\ \mathsf{true} \equiv$$
$$\underline{(\lambda xyz.zxy)E_1(\mathsf{pair}\ E_2(\mathsf{pair}\ E_3E_4))\mathsf{false}}\ \mathsf{false}\ \mathsf{true} \twoheadrightarrow_\beta^+$$
$$\mathsf{false}\ E_1(\mathsf{pair}\ E_2(\mathsf{pair}\ E_3E_4))\mathsf{false}\ \mathsf{true} \twoheadrightarrow_\beta^+$$
$$\mathsf{pair}\ E_2(\mathsf{pair}\ E_3E_4)\mathsf{false}\ \mathsf{true} \equiv$$

$\underline{(\lambda xyz.zxy)E_2(\mathsf{pair}\ E_3E_4)\mathsf{false}}\ \mathsf{true}\ \rightarrow_\beta^+$
$\mathsf{false}\ E_2(\mathsf{pair}\ E_3E_4)\mathsf{true}\ \rightarrow_\beta^+$
$\mathsf{pair}\ E_3E_4\ \mathsf{true} \equiv$
$\underline{(\lambda xyz.zxy)E_3E_4\ \mathsf{true}}\ \rightarrow_\beta^+$
$\mathsf{true}\ E_3E_4\ \rightarrow_\beta^+\ E_3.$ □

Az n-es egy másik egyszerű implementációja a következő.

Az $\langle E_1, E_2, \ldots, E_n\rangle$-nel jelölt n-es konstruktor λ-kifejezése

$$\mathsf{n\text{-}es} \equiv \lambda\vec{x}z.z\vec{x},$$

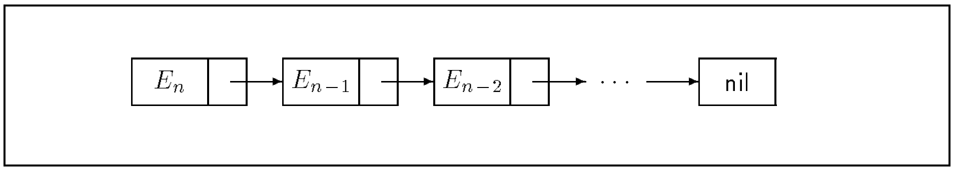
tehát az $\langle E_1, E_2, \ldots, E_n\rangle$ n-es a $\lambda z.z\vec{E}$ λ-kifejezéssel írható le. Az n-es i-edik komponensét kiválasztó szelektor λ-kifejezése

$$\mathsf{select_i} \equiv \lambda x.x(\lambda\vec{x}.x_i) \qquad (1 \leq i \leq n).$$

1.4.7. Példa. *(Az $\langle E_1, E_2, E_3, E_4\rangle$ négyes harmadik komponense)*
A négyes λ-kifejezése $\lambda z.zE_1E_2E_3E_4$, és a harmadik komponenst kiválasztó szelektor $\lambda x.x(\lambda x_1x_2x_3x_4.x_3)$. Így

$\underline{(\lambda x.x(\lambda x_1x_2x_3x_4.x_3))(\lambda z.zE_1E_2E_3E_4)}\ \rightarrow_\beta$
$\underline{(\lambda z.zE_1E_2E_3E_4)(\lambda x_1x_2x_3x_4.x_3)}\ \rightarrow_\beta$
$(\lambda x_1x_2x_3x_4.x_3)E_1E_2E_3E_4\ \rightarrow_\beta^+\ E_3.$ □

1.4.3. Listák és listákon értelmezett műveletek



1.10. ábra. A láncolt lista adatszerkezet

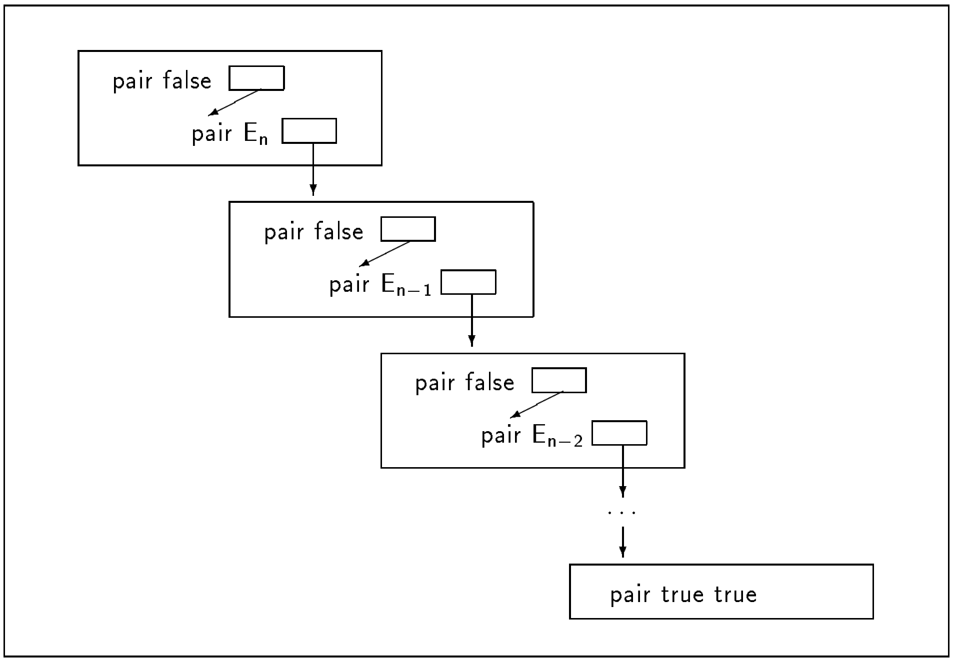
Az $\{E_n, E_{n-1}, \ldots, E_1\}$ *láncolt lista* adatszerkezetben, amelynek felépítése az 1.10. ábrán látható, E_n a lista feje, és az E_1 elem után a *nil* jelöli a lista végét. Az E_{i+1}, E_i $(1 \leq i \leq n-1)$ elemkapcsolatot jelöljük a

$\mathsf{cons} \equiv \lambda xy.\mathsf{pair\ false}(\mathsf{pair}\ xy)$

konstruktor λ-kifejezés **cons** $E_{i+1}E_i$ applikációjával, és a *nil* elem legyen

$\mathsf{nil} \equiv \mathsf{pair\ true\ true}.$

A **nil** λ-kifejezésben az utolsó λ-kifejezés egy tetszőleges λ-kifejezés is lehet, és majd később a **nil**-re egy egyszerűbb λ-kifejezést is adunk.



1.11. ábra. A láncolt lista adatszerkezet implementációja

Ez tehát azt jelenti, hogy a listát „dupla párokkal" implementáljuk, a lista minden eleméhez egy dupla pár tartozik (1.11. ábra):

- az első pár első eleme jelzi a lista végét, a *nil* elemben ez **true**, a többi elemben false,

- a második pár első eleme tartalmazza a lista elemét.

Egy új elemnek a láncolt listába beszúrása azt jelenti, hogy az új elemhez láncoljuk a már meglevő listát. Így a lista konstruktor első argumentuma mindig a beszúrandó elem, a második pedig a bővítendő lista.

Felhasználva azt, hogy a láncolt listát lényegében rendezett párokkal implementáljuk, a lista fejelemét és a maradék részét meghatározó λ-kifejezés a következő:

$\mathsf{head} \equiv \lambda x.\mathsf{first}(\mathsf{second}\ x)$,

$\mathsf{tail}\ \ \equiv \lambda x.\mathsf{second}(\mathsf{second}\ x)$.

1.4.8. Példa. *(Az $E''' = \{E_3, E_2, E_1\}$ lista fejeleme és maradék része)*
Ha a lista konstruktor λ-kifejezését felhasználva

$E' \equiv \mathsf{cons}\ E_1\ \mathsf{nil} \equiv \mathsf{pair}\ \mathsf{false}(\mathsf{pair}\ E_1\ \mathsf{nil})$,

$E'' \equiv \mathsf{cons}\ E_2 E' \equiv \mathsf{pair}\ \mathsf{false}(\mathsf{pair}\ E_2 E')$,

akkor az $\{E_3, E_2, E_1\}$ lista λ-kifejezése

$E''' \equiv \mathsf{pair}\ \mathsf{false}(\mathsf{pair}\ E_3 E'')$.

Az E''' első eleme

$\underline{(\lambda x.\mathsf{first}(\mathsf{second}\ x))E'''} \ \rightarrow_\beta$
$\mathsf{first}(\mathsf{second}\ E''') \equiv$
$\mathsf{first}(\mathsf{second}(\mathsf{pair}\ \mathsf{false}(\mathsf{pair}\ E_3 E''))) \ \rightarrow_\beta^+$
$\mathsf{first}(\mathsf{pair}\ E_3 E'') \ \rightarrow_\beta^+ \ E_3$.

Az E'''-ból a fejelem utáni maradék rész a következő:

$\underline{(\lambda x.\mathsf{second}(\mathsf{second}\ x))E'''} \ \rightarrow_\beta$
$\mathsf{second}(\mathsf{second}\ E''') \ \rightarrow_\beta^+$
$\mathsf{second}(\mathsf{pair}\ E_3 E'') \ \rightarrow_\beta^+ \ E''$. □

A rendezett párok első eleme arra szolgál, hogy egy listáról könnyen eldönthető legyen, vajon üres-e. Erre a vizsgálatra az

$\mathsf{empty} \equiv \mathsf{first}$

λ-kifejezés adható meg, hiszen egyedül csak a nil elemnél true a listát alkotó rendezett pár első tagja. Mivel a nem nil listáknál a cons konstruktor a listákat alkotó párok első elemeként a false λ-kifejezést adja meg, az empty definíciója miatt a nil üres listára bármelyik olyan λ-kifejezés megfelel, amelyik a first függvényre a true értéket adja. Egy ilyen egyszerű λ-kifejezés például a következő:

$\mathsf{nil} \equiv \lambda x.\mathsf{true}.$

1.4.4. A számkonstansok és aritmetikai műveletek

A $\mathsf{c}_0, \mathsf{c}_1, \ldots$ λ-kifejezéseket *számjegyrendszernek* vagy röviden számrendszernek nevezzük, ha $\mathsf{c}_\mathsf{i} \in \Lambda^0$ $(0 \le i)$, és van olyan succ és zero λ-kifejezés, amelyre

$\mathsf{succ}\ \mathsf{c}_\mathsf{i} \equiv \mathsf{c}_{\mathsf{i}+1} \quad (0 \le i),$

és

$$\mathsf{zero}\ \mathsf{c}_\mathsf{i} = \begin{cases} \mathsf{true}, & \text{ha } i = 0, \\ \mathsf{false}, & \text{egyébként.} \end{cases}$$

A számjegyrendszer *normált*, ha minden c_i normál formában van. A számjegyrendszer *adekvát*, ha megadható egy olyan pred λ-kifejezés, amelyre

$$\mathsf{pred}\ \mathsf{c}_\mathsf{i} = \begin{cases} \mathsf{c}_{\mathsf{i}-1}, & \text{ha } i \ge 1, \\ \mathsf{false}, & \text{egyébként.} \end{cases}$$

A $\mathsf{c}_0, \mathsf{c}_1, \ldots$ λ-kifejezéseknek a természetes számokat feleltetjük meg, az $i \in \mathbb{N}$ számot a c_i, vagy még kifejezőbben jelölve, az $\ulcorner i \urcorner$ λ-kifejezés képviseli. Három normált és adekvát számjegyrendszert mutatunk be. Az első kettőben, a harmadikhoz viszonyítva, különösen a pred függvény egyszerűsége tűnik fel, a harmadik számjegyrendszerben pedig néhány aritmetikai műveletet is definiálunk.

1. Egy egyszerű számjegyrendszer

Legyen

$$\begin{aligned}
&\ulcorner 0 \urcorner \equiv \lambda x.x \equiv \mathsf{I},\\
&\mathsf{succ} \equiv \lambda y.\mathsf{pair}\ \mathsf{false}\ y \twoheadrightarrow_\beta^+ \lambda y.(\lambda x.x\ \mathsf{false}\ y) \equiv \lambda yx.x\ \mathsf{false}\ y,\\
&\mathsf{zero} \equiv \lambda x.x\ \mathsf{true},\\
&\mathsf{pred} \equiv \lambda x.x\ \mathsf{false}.
\end{aligned}$$

1.4.9. Példa. *(Ebben a számjegyrendszerben a számok tehát a következők)*

$$\begin{aligned}
&\ulcorner 0 \urcorner \equiv && \lambda x.x,\\
&\ulcorner 1 \urcorner \equiv \mathsf{succ}\ \ulcorner 0 \urcorner \equiv (\lambda y.\mathsf{pair}\ \mathsf{false}\ y)\ulcorner 0 \urcorner \twoheadrightarrow_\beta^+ && \lambda x.x\ \mathsf{false}\ \ulcorner 0 \urcorner,\\
&\ulcorner 2 \urcorner \equiv \mathsf{succ}\ \ulcorner 1 \urcorner \equiv (\lambda y.\mathsf{pair}\ \mathsf{false}\ y)\ulcorner 1 \urcorner \twoheadrightarrow_\beta^+ && \lambda x.x\ \mathsf{false}\ \ulcorner 1 \urcorner,\\
&\ulcorner 3 \urcorner \equiv \mathsf{succ}\ \ulcorner 2 \urcorner \equiv (\lambda y.\mathsf{pair}\ \mathsf{false}\ y)\ulcorner 2 \urcorner \twoheadrightarrow_\beta^+ && \lambda x.x\ \mathsf{false}\ \ulcorner 2 \urcorner,\\
&\ldots && \ldots
\end{aligned}$$

□

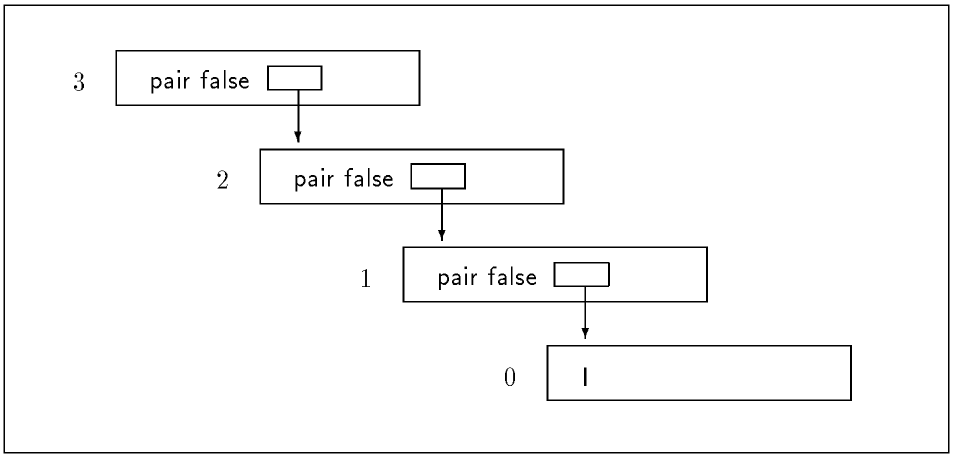
Az egyszerű számjegyrendszer számjegyeinek felépítése az 1.12. ábrán látható. A számok tehát

$$[\mathsf{false}, \mathsf{false}, \ldots, \mathsf{false}, \mathsf{I}] \equiv [\mathsf{false}, [\mathsf{false}, \ldots, [\mathsf{false}, \mathsf{I}] \ldots]]$$

típusú kifejezések. A számok szerkezetéből azonnal látható, hogy $\mathsf{zero} \equiv \mathsf{first}$, és $\mathsf{pred} \equiv \mathsf{second}$. A $\ulcorner 0 \urcorner$-nek azért célszerű az I-t választani, hogy $\mathsf{zero}\ \ulcorner 0 \urcorner = \mathsf{true}$ legyen.

1.4.10. Példa. *($\mathsf{zero}\ \ulcorner 0 \urcorner = \mathsf{true}$ és $\mathsf{zero}\ \ulcorner 1 \urcorner = \mathsf{false}$)*

$$\begin{aligned}
&\mathsf{zero}\ \ulcorner 0 \urcorner \equiv\\
&\underline{(\lambda x.x\ \mathsf{true})\ulcorner 0 \urcorner} \twoheadrightarrow_\beta\\
&\ulcorner 0 \urcorner\ \mathsf{true} \equiv\\
&\underline{(\lambda x.x)\mathsf{true}} \twoheadrightarrow_\beta \mathsf{true}.
\end{aligned}$$



1.12. ábra. Az egyszerű számjegyrendszer számainak implementációja

$\mathsf{zero}\ \ulcorner 1 \urcorner \equiv$
$\underline{(\lambda x.x\ \mathsf{true})\ulcorner 1 \urcorner} \ \to_\beta$
$\ulcorner 1 \urcorner\ \mathsf{true} \equiv$
$\underline{(\lambda x.x\ \mathsf{false}\ \ulcorner 0 \urcorner)\mathsf{true}} \ \to_\beta$
$\mathsf{true}\ \mathsf{false}\ \ulcorner 0 \urcorner \equiv$
$\underline{(\lambda xy.x)\mathsf{false}}\ \ulcorner 0 \urcorner \ \to_\beta$
$\underline{(\lambda y.\mathsf{false})\ulcorner 0 \urcorner} \ \to_\beta\ \mathsf{false}.$ □

1.4.11. Példa. *($\mathsf{pred}\ \ulcorner 2 \urcorner = \ulcorner 1 \urcorner$ és $\mathsf{pred}\ \ulcorner 0 \urcorner = \mathsf{false}$)*
$\mathsf{pred}\ \ulcorner 2 \urcorner \equiv$
$\underline{(\lambda x.x\ \mathsf{false})\ulcorner 2 \urcorner} \ \to_\beta$
$\ulcorner 2 \urcorner\ \mathsf{false} \equiv$
$\underline{(\lambda x.x\ \mathsf{false}\ \ulcorner 1 \urcorner)\mathsf{false}} \ \to_\beta$
$\mathsf{false}\ \mathsf{false}\ \ulcorner 1 \urcorner \equiv$
$\underline{(\lambda xy.y)\mathsf{false}}\ \ulcorner 1 \urcorner \ \to_\beta$
$\underline{(\lambda y.y)\ulcorner 1 \urcorner} \ \to_\beta\ \ulcorner 1 \urcorner.$

$\mathsf{pred}\ \ulcorner 0 \urcorner \equiv$

$\underline{(\lambda x.x\ \mathsf{false})\ulcorner 0 \urcorner} \twoheadrightarrow_\beta$

$\ulcorner 0 \urcorner\ \mathsf{false} \equiv$

$\underline{(\lambda x.x)\mathsf{false}} \twoheadrightarrow_\beta \mathsf{false}.$ □

2. Scott-számjegyek

Legyen

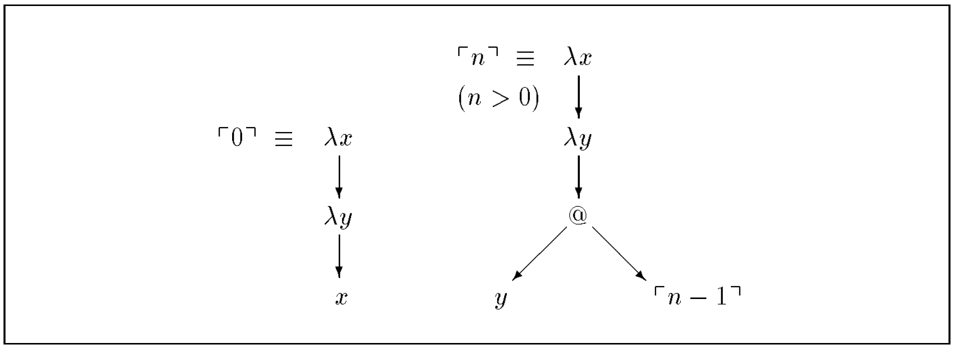
$\ulcorner 0 \urcorner \equiv \lambda xy.x \equiv \mathsf{true},$

$\mathsf{succ} \equiv \lambda zxy.yz,$

$\mathsf{zero} \equiv \lambda x.x\ \mathsf{true}(\lambda y.\mathsf{false}),$

$\mathsf{pred} \equiv \lambda x.x\ \ulcorner 0 \urcorner(\lambda y.y) \equiv \lambda x.x\ \ulcorner 0 \urcorner\ \mathsf{I}.$

Ezt az adekvát számjegyrendszert *Scott-számjegyrendszernek*, az $\ulcorner i \urcorner$ számjegyeket *Scott-számjegyeknek*, vagy röviden *Scott-számoknak* nevezzük. A Scott-számjegyek gráfja az 1.13. ábrán látható. Az ábrából kiolvasható a zero és a pred függvény működése is.



1.13. ábra. A Scott-számjegyrendszer számai

1.4.12. Példa. *(A Scott-számjegyek a következők)*

$$
\begin{array}{ll}
\ulcorner 0 \urcorner \equiv & \lambda xy.x, \\
\ulcorner 1 \urcorner \equiv \mathsf{succ}\ \ulcorner 0 \urcorner \equiv \underline{(\lambda zxy.yz)\ulcorner 0 \urcorner} \twoheadrightarrow_\beta & \lambda xy.y\ \ulcorner 0 \urcorner, \\
\ulcorner 2 \urcorner \equiv \mathsf{succ}\ \ulcorner 1 \urcorner \equiv \underline{(\lambda zyx.yz)\ulcorner 1 \urcorner} \twoheadrightarrow_\beta & \lambda xy.y\ \ulcorner 1 \urcorner, \\
\ulcorner 3 \urcorner \equiv \mathsf{succ}\ \ulcorner 2 \urcorner \equiv \underline{(\lambda zyx.yz)\ulcorner 2 \urcorner} \twoheadrightarrow_\beta & \lambda xy.y\ \ulcorner 2 \urcorner, \\
\ldots & \ldots
\end{array}
$$

□

Végezzük el az előző számjegyrendszer példáit a Scott-számokra is.

1.4.13. Példa. *($\mathsf{zero}\ \ulcorner 0 \urcorner = \mathsf{true}$ és $\mathsf{zero}\ \ulcorner 1 \urcorner = \mathsf{false}$)*

$$
\begin{array}{l}
\mathsf{zero}\ \ulcorner 0 \urcorner \equiv \\
\underline{(\lambda x.x\ \mathsf{true}(\lambda y.\mathsf{false}))\ulcorner 0 \urcorner} \twoheadrightarrow_\beta \\
\ulcorner 0 \urcorner\ \mathsf{true}(\lambda y.\mathsf{false}) \equiv \\
\underline{(\lambda xy.x)\mathsf{true}}(\lambda y.\mathsf{false}) \twoheadrightarrow_\beta \\
\underline{(\lambda y.\mathsf{true})(\lambda y.\mathsf{false})} \twoheadrightarrow_\beta \\
\mathsf{true}.
\end{array}
$$

$$
\begin{array}{l}
\mathsf{zero}\ \ulcorner 1 \urcorner \equiv \\
\underline{(\lambda x.x\ \mathsf{true}(\lambda y.\mathsf{false}))\ulcorner 1 \urcorner} \twoheadrightarrow_\beta \\
\ulcorner 1 \urcorner\ \mathsf{true}(\lambda y.\mathsf{false}) \equiv \\
\underline{(\lambda xy.y\ \ulcorner 0 \urcorner)\mathsf{true}}(\lambda y.\mathsf{false}) \twoheadrightarrow_\beta \\
\underline{(\lambda y.y\ \ulcorner 0 \urcorner)(\lambda y.\mathsf{false})} \twoheadrightarrow_\beta \\
\underline{(\lambda y.\mathsf{false})\ulcorner 0 \urcorner} \twoheadrightarrow_\beta \\
\mathsf{false}.
\end{array}
$$

□

1.4.14. Példa. *($\mathsf{pred}\ \ulcorner 2 \urcorner = \ulcorner 1 \urcorner$ és $\mathsf{pred}\ \ulcorner 0 \urcorner = \ulcorner 0 \urcorner$)*

$$
\begin{array}{l}
\mathsf{pred}\ \ulcorner 2 \urcorner \equiv \\
\underline{(\lambda x.x\ \ulcorner 0 \urcorner(\lambda y.y))\ulcorner 2 \urcorner} \twoheadrightarrow_\beta \\
\ulcorner 2 \urcorner \ulcorner 0 \urcorner(\lambda y.y) \equiv \\
\underline{(\lambda xy.y\ \ulcorner 1 \urcorner)\ulcorner 0 \urcorner}(\lambda y.y) \twoheadrightarrow_\beta \\
\underline{(\lambda y.y\ \ulcorner 1 \urcorner)(\lambda y.y)} \twoheadrightarrow_\beta
\end{array}
$$

$\underline{(\lambda y.y)\ulcorner 1\urcorner} \rightarrow_\beta \ulcorner 1\urcorner.$

$\mathsf{pred}\ \ulcorner 0\urcorner \equiv$
$\underline{(\lambda x.x\ \ulcorner 0\urcorner(\lambda y.y))\ulcorner 0\urcorner} \rightarrow_\beta$
$\ulcorner 0\urcorner\ulcorner 0\urcorner(\lambda y.y) \equiv$
$\underline{(\lambda xy.x)\ulcorner 0\urcorner}(\lambda y.y) \rightarrow_\beta$
$\underline{(\lambda y.\ulcorner 0\urcorner)(\lambda y.y)} \rightarrow_\beta \ulcorner 0\urcorner.$ □

A Scott-számokra szokás egy más stílusú zero' függvényt is adni:

$\mathsf{zero}' \equiv \lambda xyz.xyz.$

Ez a függvény a $\ulcorner 0\urcorner$ Scott-számjegyre true értéket ad. Három argumentum esetén, ha az első argumentum a $\ulcorner 0\urcorner$ Scott-számjegy, akkor a függvény a második argumentumot adja eredményül, ha az első argumentum egy nem nulla $\ulcorner i\urcorner$ Scott-számjegy, akkor az eredmény a harmadik argumentummal és $\ulcorner i-1\urcorner$-gyel képzett applikáció:

$$\mathsf{zero}'\ \ulcorner i\urcorner\ uv \rightarrow_\beta^+ \begin{cases} u, & \text{ha } \ulcorner i\urcorner \equiv \ulcorner 0\urcorner, \\ v\ \ulcorner i-1\urcorner, & \text{egyébként.} \end{cases}$$

1.4.15. Példa. *($\mathsf{zero}'\ \ulcorner 0\urcorner = \lambda yz.y \equiv \mathsf{true}$ és $\mathsf{zero}'\ \ulcorner 1\urcorner = \lambda yz.z\ulcorner 0\urcorner$)*

$\mathsf{zero}'\ \ulcorner 0\urcorner \equiv$
$\underline{(\lambda xyz.xyz)\ulcorner 0\urcorner} \rightarrow_\beta$
$\lambda yz.\ulcorner 0\urcorner\ yz \equiv$
$\lambda yz.(\lambda xy.x)yz \leftrightarrow_\alpha$
$\lambda yz.\underline{(\lambda xv.x)y}z \rightarrow_\beta$
$\lambda yz.\underline{(\lambda v.y)z} \rightarrow_\beta$
$\lambda yz.y \equiv \mathsf{true}.$

$\mathsf{zero}'\ \ulcorner 1\urcorner \equiv$
$\underline{(\lambda xyz.xyz)\ulcorner 1\urcorner} \rightarrow_\beta$
$\lambda yz.\ulcorner 1\urcorner\ yz \equiv$

$\lambda yz.\underline{(\lambda xy.y\ulcorner 0\urcorner)yz} \twoheadrightarrow_\beta$
$\lambda yz.\underline{(\lambda y.y\ulcorner 0\urcorner)z} \twoheadrightarrow_\beta$
$\lambda yz.z\ulcorner 0\urcorner.$

A kapott eredményekből a fentiekben a zero' függvényre tett megjegyzés jól látható. □

3. Church-számok

A 1.4.2. pontban leírt $f^n(x)$ jelölést használva, legyen

$$\mathsf{c_n} \equiv \lambda fx.f^n(x) \qquad (n = 0, 1, 2, \dots)$$

A $\mathsf{c_0}, \mathsf{c_1}, \mathsf{c_2}, \dots$ λ-kifejezéseket *Church-számjegyeknek*, vagy röviden *Church-számoknak* nevezzük.

Egy Church-szám rákövetkezőjét a

$$\mathsf{succ} \equiv \lambda nfx.f(nfx)$$

λ-kifejezés állítja elő.

1.4.16. Példa. ($\mathsf{succ}\ c_0 \equiv \mathsf{c_1}$)
$\mathsf{succ}\ \mathsf{c_0} \equiv \underline{(\lambda xyz.y(xyz))\mathsf{c_0}} \twoheadrightarrow_\beta$
$\lambda yz.y(\mathsf{c_0}yz) \equiv \lambda yz.y(\underline{(\lambda fx.x)yz}) \twoheadrightarrow_\beta$
$\lambda yz.y(\underline{(\lambda x.x)z}) \twoheadrightarrow_\beta$
$\lambda yz.yz \leftrightarrow_\alpha \lambda fx.fx \equiv \mathsf{c_1}.$ □

1.4.17. Példa. *(A Church-számok a következők)*
$\mathsf{c_0} \equiv \ulcorner 0\urcorner \equiv \qquad \lambda fx.x,$
$\mathsf{c_1} \equiv \ulcorner 1\urcorner \equiv \mathsf{succ}\ \ulcorner 0\urcorner \equiv \underline{(\lambda nfx.f(nfx)\ulcorner 0\urcorner} \twoheadrightarrow_\beta^+ \lambda fx.f(x),$
$\mathsf{c_2} \equiv \ulcorner 2\urcorner \equiv \mathsf{succ}\ \ulcorner 1\urcorner \equiv \underline{(\lambda nfx.f(nfx)\ulcorner 1\urcorner} \twoheadrightarrow_\beta^+ \lambda fx.f(f(x)),$
$\mathsf{c_3} \equiv \ulcorner 3\urcorner \equiv \mathsf{succ}\ \ulcorner 2\urcorner \equiv \underline{(\lambda nfx.f(nfx)\ulcorner 2\urcorner} \twoheadrightarrow_\beta^+ \lambda fx.f(f(f(x))),$
$\dots \qquad \dots$ □

Megjegyezzük, hogy az n természetes számnak megfelelő $\mathsf{c_n}$ λ-kifejezés azt jelenti, hogy egy $\mathsf{c_n} EF$ applikációban az első aktuális paraméter kifejezését éppen n-szer kell alkalmazni a második aktuális paraméterre:

$$\mathsf{c_n} EF \twoheadrightarrow_\beta^+ \underbrace{E(E(\ldots E(E}_{n} F)\ldots)),$$

ezért a Church-számokat *iterátoroknak* is nevezhetjük.

Könnyen belátható, hogy egy Church-szám rákövetkezőjét a

$$\mathsf{succ}' \equiv \lambda nfx.nf(fx)$$

λ-kifejezés is előállítja, és az is azonnal látszik, hogy $\mathsf{succ} \neq \mathsf{succ}'$.

Definiálható a **zero** függvény is, amely a $\mathsf{c_0}$ argumentumra a **true**, minden más argumentumra a **false** értéket adja:

$$\mathsf{zero} \equiv \lambda x.x(\mathsf{true\ false})\ \mathsf{true} \equiv \lambda x.x(\lambda y.\mathsf{false})\mathsf{true}.$$

Látható, hogy a $\mathsf{zero\ c_n}$ kifejezésben az első argumentum egy olyan függvény, amelyik minden paraméterre a **false** értéket adja, a második argumentum pedig a **true** érték. Ha $\mathsf{c_n}$ nulla, akkor a fenti megjegyzés szerint az első argumentumot nullaszor kell alkalmazni a második argumentumra, azaz az eredmény a második argumentum, ha $\mathsf{c_n}$ nem nulla, akkor az első argumentumot n-szer kell alkalmazni a második argumentumra, de az első argumentum mindig a **false** értéket adja eredményül.

1.4.18. Példa. *(*$\mathsf{zero\ c_0} = \mathsf{true}$ *és* $\mathsf{zero\ c_2} = \mathsf{false}$ *)*

$\mathsf{zero\ c_0} \equiv$
$\underline{(\lambda x.x(\mathsf{true\ false})\ \mathsf{true})\mathsf{c_0}} \twoheadrightarrow_\beta$
$\mathsf{c_0}(\mathsf{true\ false})\mathsf{true} \equiv$
$\underline{(\lambda fx.x)(\mathsf{true\ false})}\mathsf{true} \twoheadrightarrow_\beta$
$\underline{(\lambda x.x)\mathsf{true}} \twoheadrightarrow_\beta \mathsf{true}.$

$\mathsf{zero\ c_2} \equiv \underline{(\lambda x.x(\mathsf{true\ false})\mathsf{true})\mathsf{c_2}} \twoheadrightarrow_\beta$
$\mathsf{c_2}(\mathsf{true\ false})\mathsf{true} \equiv$
$\underline{(\lambda fx.f(fx))(\mathsf{true\ false})}\mathsf{true} \twoheadrightarrow_\beta$

$\underline{(\lambda x.(\mathsf{true\ false})((\mathsf{true\ false})x))\mathsf{true}} \ \twoheadrightarrow_\beta$
$(\mathsf{true\ false})((\mathsf{true\ false})\mathsf{true}) \equiv$
$(\underline{(\lambda xy.x)\ \mathsf{false}})((\mathsf{true\ false})\mathsf{true}) \ \twoheadrightarrow_\beta$
$\underline{(\lambda y.\mathsf{false})((\mathsf{true\ false})\mathsf{true})} \ \twoheadrightarrow_\beta \ \mathsf{false}.$ □

A nem nulla Church-szám megelőzőjét meghatározó λ-kifejezés bonyolultabb, erre most négy megoldást is adunk.

1. $\mathsf{pred}' \equiv \lambda n.(n(\lambda yz.z(\mathsf{succ}(y\ \mathsf{true}))(y\ \mathsf{true}))(\lambda z.z\ \mathsf{c_0 c_0}))\mathsf{false}$,

2. $\mathsf{pred}'' \equiv \lambda n.\mathsf{second}(n(\lambda y.\mathsf{pair\ succ}((\mathsf{first}\ y)(\mathsf{second}\ y)))(\mathsf{pair}(\lambda z.\mathsf{c_0})\mathsf{c_0}))$,

3. $\mathsf{pred}''' \equiv \lambda n.(\lambda fx.\mathsf{second}(n(\mathsf{pref}'''\ f)(\mathsf{pair\ true}\ x)))$,

ahol

$\mathsf{pref}''' \equiv \lambda fp.\mathsf{pair\ false}(\mathsf{if}(\mathsf{first}\ p)(\mathsf{second}\ p)(f(\mathsf{second}\ p)))$.

Ennek a függvénynek a működése könnyen átlátható, ha észrevesszük azt, hogy

$\mathsf{pref}'''\ f(\mathsf{pair}\ bx) \ \twoheadrightarrow_\beta^+ \ \mathsf{pair\ false}(\mathsf{if}(b)(x)(fx))$,

és így

$\mathsf{pref}'''\ f(\mathsf{pair\ true}\ x) \ \twoheadrightarrow_\beta^+ \ \mathsf{pair\ false}\ x$.

Mivel

$\mathsf{pref}'''\ f(\mathsf{pair\ false}\ x) \ \twoheadrightarrow_\beta^+ \ \mathsf{pair\ false}(fx)$,
$(\mathsf{pref}'''\ f)^k(\mathsf{pair\ false}\ x) \ \twoheadrightarrow_\beta^+ \ \mathsf{pair\ false}(f^k(x)) \quad (k > 0)$,

ezért

$\mathsf{c_n}(\mathsf{pref}'''\ f)(\mathsf{pair\ false}\ x) \ \twoheadrightarrow_\beta^+$
$(\mathsf{pref}'''\ f)^n(\mathsf{pair\ true}\ x) \ \twoheadrightarrow_\beta^+$
$\mathsf{pair\ false}(f^{n-1}(x)) \quad (n > 0)$.

Ezt felhasználva

$\mathsf{pred}''' \; \mathsf{c_n} \; fx \; \rightarrow^{+}_{\beta} \; f^{n-1}(x),$

azaz

$\mathsf{pred}''' \; \mathsf{c_n} \; \rightarrow^{+}_{\beta} \; \lambda fx.f^{n-1}(x) \equiv \mathsf{c_{n-1}}.$

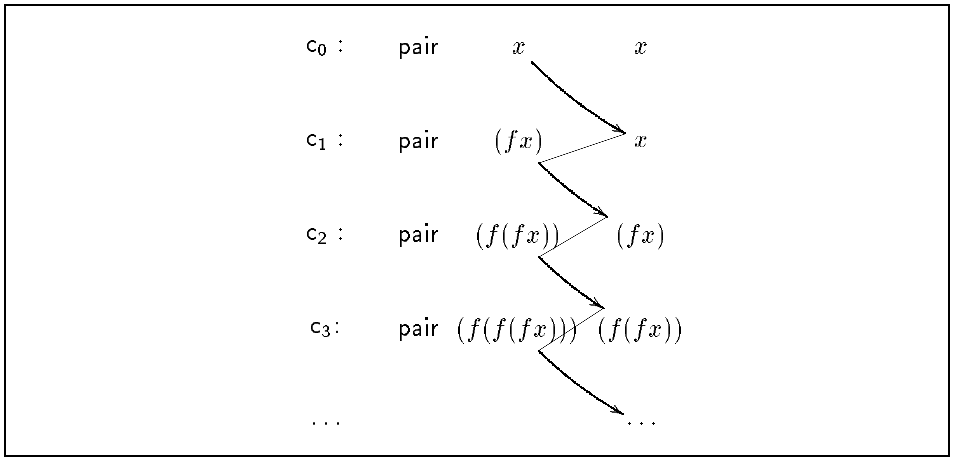
4. Az előző pontban megadott függvény egy kicsit egyszerűbben is megadható, a megelőző számjegy meghatározásának elve hasonló:

$\mathsf{pred}^{IV} \equiv \lambda n.(\lambda fx.\mathsf{second}(n(\mathsf{pref}^{IV} \; f)(\mathsf{pair} \; xx))),$

ahol

$\mathsf{pref}^{IV} \equiv \lambda fp.\mathsf{pair}(f(\mathsf{first} \; p))(\mathsf{first} \; p).$

A $\mathsf{c_n}(\mathsf{pref}^{IV} \; f)(\mathsf{pair} \; xx)$ kifejezés egy olyan párt állít elő, amelyiknek a második tagja éppen a $\mathsf{c_{n-1}}$ szám λ-absztrakciójának a törzse. A pref^{IV} kifejezés minden alkalmazásánál „átmásolja” az argumentumként megadott pár első tagját az új pár második tagjába, és az első tagban az f applikációinak számát eggyel növeli (1.14. ábra).



1.14. ábra. A $\mathsf{pref} \; f(\mathsf{pair} \; xx)$ értékei

Látható tehát, hogy a Church-számjegyrendszer egy adekvát számjegyrendszer.

A Church-számokra Rosser definiálta az összeadás, szorzás és hatványozás műveleteket:

$\mathsf{add} \equiv \lambda xypq.xp(ypq)$,
$\mathsf{mul} \equiv \lambda xyp.x(yp)$,
$\mathsf{exp} \equiv \lambda xy.yx$.

Könnyen belátható, hogy a természetes számoknak megfeleltetett Church-számokra ezek a műveletek éppen a megfelelő aritmetikai művelet eredményének megfeleltetett Church-számot adják.

1.4.19. Példa. *(A $\mathsf{c_i}$ és a $\mathsf{c_j}$ összeadásának eredménye $\mathsf{c_{i+j}}$)*

$\mathsf{add}\ \mathsf{c_i c_j} \equiv \underline{(\lambda xypq.xp(ypq))\mathsf{c_i}}\mathsf{c}_j \ \rightarrow_\beta$
$\underline{(\lambda ypq.\mathsf{c_i}p(ypq))\mathsf{c_j}} \ \rightarrow_\beta$
$\lambda pq.\mathsf{c_i}p(\mathsf{c_j}pq) \equiv$
$\lambda pq.\underline{(\lambda fx.f^i(x))p}(\mathsf{c_j}pq) \ \rightarrow_\beta$
$\lambda pq.\underline{(\lambda x.p^i(x))(\mathsf{c_j}pq)} \ \rightarrow_\beta$
$\lambda pq.p^i(\mathsf{c_j}pq) \equiv$
$\lambda pq.p^i(\underline{(\lambda fx.f^j(x))p}q) \ \rightarrow_\beta$
$\lambda pq.p^i(\underline{(\lambda x.p^j(x))q}) \ \rightarrow_\beta$
$\lambda pq.p^i(p^j(q)) \ \leftrightarrow_\alpha$
$\lambda fx.f^i(f^j(x)) \equiv$
$\lambda fx.f^{i+j}(x) \equiv \mathsf{c_{i+j}}$. □

1.4.20. Példa. *(A $\mathsf{c_i}$ és a $\mathsf{c_j}$ szorzásának eredménye $\mathsf{c_{i*j}}$)*

$\mathsf{mul}\ \mathsf{c_i c_j} \equiv \underline{(\lambda xyp.x(yp))\mathsf{c_i}}\mathsf{c_j} \ \rightarrow_\beta$
$\underline{(\lambda yp.\mathsf{c_i}(yp))\mathsf{c_j}} \ \rightarrow_\beta$
$\lambda p.\mathsf{c_i}(\mathsf{c_j}p) \equiv$
$\lambda p.\mathsf{c_i}(\underline{(\lambda fx.f^j(x))p}) \ \rightarrow_\beta$

$$\begin{aligned}
&\lambda p.\mathsf{c}_\mathsf{i}(\lambda x.p^j(x)) \equiv \\
&\lambda p.\underline{(\lambda fx.f^i(x))(\lambda x.p^j(x))} \;\leftrightarrow_\alpha \\
&\lambda p.\underline{(\lambda fx.f^i(x))(\lambda z.p^j(z))} \;\rightarrow_\beta \\
&\lambda p.(\lambda x.(\lambda z.p^j(z))^i(x)) \;\leftrightarrow_\alpha \\
&\lambda f.(\lambda x.(\lambda z.f^j(z))^i(x)) \equiv \lambda fx.(\lambda z.f^j(z))^i(x).
\end{aligned}$$

Már csak azt kell belátni, hogy

$$(\lambda z.f^j(z))^i(x) \;\rightarrow_\beta^+\; f^{i*j}(x) \equiv f^j(\ldots(f^j(f^j(x)))\ldots).$$

Ez viszont nyilvánvaló, hiszen

$$\begin{aligned}
&(\lambda z.f^j(z))^i(x) \equiv \\
&((\lambda z.f^j(z))(\ldots((\lambda z.f^j(z))(\underline{(\lambda z.f^j(z))\;(x)}))\ldots)) \;\rightarrow_\beta \\
&((\lambda z.f^j(z))(\ldots(\underline{(\lambda z.f^j(z))(f^j(x))})\ldots)) \;\rightarrow_\beta \\
&((\lambda z.f^j(z))(\ldots(f^j(f^j(x)))\ldots)) \;\rightarrow_\beta^+ \\
&f^j(\ldots(f^j(f^j(x)))\ldots).
\end{aligned}$$

□

Az aritmetikai műveletek leírhatók a succ és pred felhasználásával is, például:

$\mathsf{add}' \equiv \lambda xy.x\;\mathsf{succ}\;y,$

$\mathsf{sub}' \equiv \lambda xy.y\;\mathsf{pred}\;x,$

a szorzás és a hatványozás az összeadás és a szorzás felhasználásával:

$\mathsf{mul}' \equiv \lambda xy.x(\mathsf{add}\;y)\mathsf{c}_0,$

$\mathsf{exp}' \equiv \lambda xy.y(\mathsf{mul}\;x)\mathsf{c}_1,$

Felhasználva a sub műveletnek azt a tulajdonságát, hogy $m \leq n$ esetén $\mathsf{sub}\;mn = \mathsf{c}_0$, leírhatjuk a két szám egyenlőségét vizsgáló λ-kifejezést is:

$\mathsf{equal} \equiv \lambda xy.\mathsf{zero}(\mathsf{add}(\mathsf{sub}\;xy)(\mathsf{sub}\;yx)),$

vagy egy másik forma:

$\mathsf{equal}' \equiv \lambda xy.\mathsf{and}(\mathsf{zero}(m\ \mathsf{pred}\ n))(\mathsf{zero}(n\ \mathsf{pred}\ m))$.

Megjegyezzük, hogy a konstansok definíciójával a λ-kalkulus egy új, nem deklarált fogalommal, a konstansok *típusával* bővül, hiszen a konstansok már logikai vagy nemnegatív egész konstansok, és a függvényapplikációkban már meg kell követelni azt, hogy a függvények argumentuma megfeleljen a függvénytől függő előre megadott típusnak.

1.4.5. A δ-redukció

A 1.4.19. és a 1.4.20. példákból is látható, hogy a definiált konstansokkal a műveletek végzése kissé nehézkes. A λ-kalkulus kifejezései könnyebben és kényelmesebben írhatók és olvashatók, ha a λ-kalkulus beépített, előre definiált konstansokat, és az ezeken a konstansokon értelmezett előre definiált függvényeket tartalmaz. Ezért a továbbiakban a számkonstansokat leíró λ-kifejezéseket a reprezentált számkonstanssal, a függvényeket a szokásos műveleti jelekkel jelölhetjük, tehát például a $\ulcorner 0 \urcorner$ vagy a c_0 helyett 0-t, az add helyett a $+$ jelet, az equal helyett az $=$ jelet is írhatjuk. A konstansokon értelmezett függvényeket *δ-függvényeknek* nevezzük. A δ-függvények nem csak a leírást egyszerűsítik, hanem a kifejezések átalakítását is gyorsítják, mivel β-redukciókból és α-konverziókból álló sorozatok végrehajtását teszik feleslegessé.

1.4.21. Definíció. A δ-redukció:

Ha egy függvényapplikációban szereplő függvény egy δ-függvény és az aktuális paraméter konstans, akkor a függvényapplikáció helyettesíthető azzal az értékkel, amelyet a függvény a paraméterrel megadott pontban felvesz.

1.4.22. Példa. *(A δ-redukció felhasználása)*

$+\ 2\ 3\ \rightarrow_\delta\ 5$,

$+\ (*\ 2\ 3)\ 4\ \rightarrow_\delta\ +\ 6\ 4\ \rightarrow_\delta\ 10$,

$\mathsf{and\ true\ false}\ \rightarrow_\delta\ \mathsf{false}$,

$=\ 3\ 4\ \rightarrow_\delta\ \mathsf{false}$. □

Mivel a δ-redukció a konstansok részhalmazán egy előre megadott műveletet végez, önmaga is konstansnak tekinthető. Ha az egyszerű típusnélküli λ-kalkulust bővítjük a konstansokkal és a konstansokon értelmezett δ-függvényekkel, azaz

$$
\begin{array}{lll}
< \textit{konstans} > & ::= & < \textit{számkonstans} > \\
 & | & < \textit{logikai konstans} > \\
 & | & < \delta\textit{-függvény} >,
\end{array}
$$

akkor *konstansos típusnélküli λ-kalkulusról* vagy röviden a *típusnélküli λ-kalkulusról* beszélünk. Az 1.1.1. definícióban meghatározott egyszerű típusnélküli λ-kifejezések tehát a következőképpen bővíthetők:

1.4.23. Definíció. A konstansos típusnélküli λ-kifejezés:

Tegyük fel, hogy adott egy nem feltétlenül véges, egymástól páronként különböző változókat (szimbólumokat), konstansokat, a λ jelet, a pontot, valamint a nyitó- és csukózárójeleket tartalmazó halmaz. Ezen a halmazon, mint ábécén értelmezett λ-kifejezések a következő szavak:

$$
\begin{array}{lll}
< \lambda\textit{-kifejezés} > & ::= & < \textit{változó} > \\
 & | & < \textit{konstans} > \\
 & | & < \lambda\textit{-absztrakció} > \\
 & | & < \textit{applikáció} > \\
< \lambda\textit{-absztrakció} > & ::= & (\lambda \ < \textit{változó} > \ . \ < \lambda\textit{-kifejezés} > \ ) \\
< \textit{applikáció} > & ::= & (\ < \lambda\textit{-kifejezés} > \ < \lambda\textit{-kifejezés} > \ )
\end{array}
$$

A konstansos típusnélküli λ-kalkulust *alkalmazott* λ-kalkulusnak is nevezhetjük.

1.5. A rekurzió

Mivel a λ-kalkulusban a függvényeknek nincs nevük, úgy tűnhet, hogy a λ-kalkulusban rekurzív függvényhívások nem lehetségesek. A fixpont-kombinátor alkalmazásával azonban a λ-kalkulus bővítése nélkül le tudunk írni rekurzív függvényeket.

A fixpont-kombinátort és alkalmazását a $fac(n) = n!$ függvényen mutatjuk be. A függvény szokásos

$$fac(n) = if\ (n = 0)\ then\ 1\ else\ (n * fac\,(n-1))$$

definícióját a λ-kalkulus jelöléseivel átírva

$$\mathsf{fac} \equiv \lambda n.\mathsf{if}(=\ n\ 0)1(*\ n(\mathsf{fac}(-\ n\ 1))),$$

ami természetesen nem helyes, mert jobboldalon nem hivatkozhatunk a fac függvényre, hiszen ekkor a fac helyettesítéseivel végtelen hosszú λ-kifejezést kapnánk. Azonban tegyük fel, hogy a jobboldalt egy

$$\mathsf{H} \equiv \lambda f.(\lambda n.\mathsf{if}(=\ n\ 0)1(*\ n(f(-\ n\ 1))))$$

függvény és a fac, mint argumentum applikációjával kaptuk, azaz

$$\mathsf{fac} = \mathsf{H}\ \mathsf{fac}.$$

Látható, hogy megszűnik a rekurzív függvényhívás, a H függvény a fac argumentumra a fac eredményt adja.

1.5.1. Definíció. A λ-kifejezés fixpontja:

Ha az E és F λ-kifejezésekre $F = EF$, akkor az F λ-kifejezést a E fixpontjának nevezzük.

A feladat tehát az, hogy megtaláljuk a H függvény fixpontját, hiszen H fixpontja lesz a fac függvény.

1.5.2. Példa. *(A λ-kifejezés fixpontjainak darabszáma)*
A $\lambda x.6$ és $(\lambda x. -\ 12\ x)$ λ-kifejezéseknek egy fixpontjuk van:

$$(\lambda x.6)6 = 6,$$
$$(\lambda x. -\ 12\ x)6 = 6.$$

Egy λ-kifejezésnek több fixpontja is lehet:

$$(\lambda x. *\ x\ x)0 \ \twoheadrightarrow_\beta\ *\ 0\ 0\ \twoheadrightarrow_\delta\ 0,$$

$(\lambda x. * x\ x)1 \twoheadrightarrow_\beta * 1\ 1 \twoheadrightarrow_\delta 1.$

A $\lambda x. * 1\ x$ λ-kifejezésnek minden $\mathsf{c_i}$ $(i \geq 0)$ számkonstans fixpontja:

$(\lambda x. * 1\ x)\ \mathsf{c_i} = \mathsf{c_i}.$

Van olyan λ-kifejezés, amelynek minden λ-kifejezés a fixpontja, minden E-re:

$(\lambda x.x)E = E.$

Nem minden λ-kifejezésnek van fixpontja, a

$(\lambda x. + x\ 1)$

λ-kifejezés semmilyen $\mathsf{c_i}$ számkonstans argumentumra sem adja $\mathsf{c_i}$-t eredményül. □

1.5.1. A fixpont-kombinátor

Tegyük fel, hogy $\mathsf{Y} \in \Lambda^0$ egy olyan λ-absztrakció, amelyet tetszőleges E λ-kifejezéssel applikálva eredményül az E λ-kifejezés fixpontját kapjuk meg, azaz

$\mathsf{Y}E = E(\mathsf{Y}E).$

Az Y tehát előállítja egy λ-kifejezés fixpontját, az ilyen tulajdonságú Y λ-absztrakciót *fixpont-kombinátornak* nevezzük. Legyen

$\mathsf{Y} \equiv \lambda x.(\lambda y.x(yy))(\lambda y.x(yy)).$

Ezt a fixpont-kombinátort Curry fedezte fel, aki ezt az Y-t *paradoxon-kombinátornak* nevezte, mivel az Y-t a Russell-paradoxonból (lásd 1.5.3.) vezette le.

Az Y-ra teljesül, hogy minden E λ-kifejezésre $\mathsf{Y}E = E(\mathsf{Y}E)$.

$\mathsf{Y}E \equiv$

$\underline{(\lambda x.(\lambda y.x(yy))(\lambda y.x(yy)))E} \twoheadrightarrow_\beta$

$\underline{(\lambda y.E(yy))(\lambda y.E(yy))} \twoheadrightarrow_\beta$
$E((\lambda y.E(yy))(\lambda y.E(yy))) \twoheadleftarrow_\beta$
$E((\lambda x.(\lambda y.x(yy))(\lambda y.x(yy)))E) \equiv$
$E(\mathsf{Y}E).$

Felhívjuk a figyelmet arra, hogy $\mathsf{Y}E = E(\mathsf{Y}E)$, de $\mathsf{Y}E \not\twoheadrightarrow_\beta^+ E(\mathsf{Y}E)$, és megjegyezzük, hogy az Y fixpont-kombinátor csak lusta paraméterkiértékeléssel használható, mohó paraméterkiértékeléssel az

$$\mathsf{Y}E = E(\underline{\mathsf{Y}E}) = E(E(\underline{\mathsf{Y}E})) = E(E(E(\underline{\mathsf{Y}E}))) = \ldots$$

végtelen sorozatot kapjuk.

Ha az $\mathsf{Y}E \equiv (\lambda x.(\lambda y.x(yy))(\lambda y.x(yy)))E$ λ-kifejezést F-fel jelöljük, akkor a fentiek alapján kimondható a következő állítás:

1.5.3. Tétel. (Az 1. fixpont tétel)

Minden E λ-kifejezéshez létezik olyan F λ-kifejezés, amelyre $F = EF$.

Egy másik fixpont-kombinátor Turingtól származik:

$$\Theta \equiv (\lambda xy.y(xxy))(\lambda xy.y(xxy)).$$

Azt kell belátnunk, hogy tetszőleges E λ-kifejezésre a ΘE applikáció előállítja az E fixpontját:

$$\Theta E = E(\Theta E).$$

A baloldalra β-redukciókat alkalmazva

$\Theta E \equiv$
$\underline{(\lambda xy.y(xxy))(\lambda xy.y(xxy))}E \twoheadrightarrow_\beta$
$\underline{(\lambda y.y((\lambda xy.y(xxy))(\lambda xy.y(xxy))y))E} \twoheadrightarrow_\beta$
$E((\lambda xy.y(xxy))(\lambda xy.y(xxy))E) \equiv$
$E(\Theta E).$

Látható, hogy a Θ fixpont-kombinátorra a $\Theta E \twoheadrightarrow_\beta^+ E(\Theta E)$ is teljesül, és így az 1. fixpont tétel $F \equiv \Theta E$ választással $F \twoheadrightarrow_{\beta}^{+} EF$ alakban is felírható.

Az 1. fixpont tétel általánosítható kettő, sőt több λ-kifejezésre is:

1.5.4. Tétel. (Dupla fixpont tétel)

Minden E és F λ-kifejezéshez létezik olyan G_1 és G_2 λ-kifejezés, amelyre $G_1 = EG_1G_2$ és $G_2 = FG_1G_2$.

Bizonyítás. Az $G_1 \equiv \Theta(\lambda u.EuG_2)$ és $G_2 \equiv \Theta(\lambda v.FG_1v)$ választással, az 1. fixpont tételt alkalmazva,

$$\begin{aligned}
&G_1 \equiv \Theta(\lambda u.EuG_2) = \\
&(\lambda u.EuG_2)(\Theta(\lambda u.EuG_2)) = \\
&E(\Theta(\lambda u.EuG_2))G_2 \equiv \\
&EG_1G_2,
\end{aligned}$$

és hasonlóan

$$\begin{aligned}
&G_2 \equiv \Theta(\lambda v.FG_1v) = \\
&(\lambda v.FG_1v)(\Theta(\lambda v.FG_1v)) = \\
&FG_1(\Theta(\lambda v.FG_1v)) \equiv \\
&FG_1G_2.
\end{aligned}$$

□

Bizonyítás nélkül közöljük a következő tételt:

1.5.5. Tétel. (Többszörös fixpont tétel)

Minden $E_1, E_2, \ldots, E_n$ λ-kifejezéshez léteznek olyan $F_1, F_2, \ldots, F_n$ λ-kifejezések, amelyekre

$$\begin{aligned}
F_1 &= E_1F_1F_2\ldots F_n, \\
F_2 &= E_2F_1F_2\ldots F_n, \\
&\ldots \\
F_n &= E_nF_1F_2\ldots F_n.
\end{aligned}$$

Az Y és a Θ fixpont-kombinátoron kívül még sok fixpont-kombinátor létezik. Néhány fixpont-kombinátor:

- *Böhm és van der Mey:* Legyen $G = \lambda xy.y(xy)$. Ekkor a G fixpontja fixpont-kombinátor, azaz ha $F = GF$, akkor minden E-re $FE = E(FE)$.
- *Böhm:* Legyen $\mathsf{Y}^0 \equiv \mathsf{Y}$, és legyen $\mathsf{Y}^{n+1} \equiv \mathsf{Y}^n(G)$. Ekkor az

  $\mathsf{Y}^0, \mathsf{Y}^1, \mathsf{Y}^2, \ldots$

  mindegyike fixpont-kombinátor.

  Böhm bizonyította be azt is, hogy $\mathsf{Y}^1 \twoheadrightarrow_\beta^+ \Theta$.
- Ha E egy tetszőleges λ-kifejezés, akkor a

  $\lambda x.(\lambda yz.x(yyz))(\lambda yz.x(yyz))E$

  λ-kifejezés az E fixpont-kombinátora.
- *Klop:* Legyen

  $£ \equiv \lambda abcdefghijklmnopqstuvwxyzr.r(thisisafixedpointcombinator),$
  $\$ \equiv ££££££££££££££££££££££££££.$

  Ekkor a \$ egy fixpont-kombinátor.
- Az előző kombinátorához hasonlóan:

  $\heartsuit \equiv \lambda ábcdefghijklmnoöprstvxyza.a(csörnyeifixpontkombinátora),$
  $\diamondsuit \equiv \heartsuit.$

  A $\diamondsuit$ egy fixpont-kombinátor.

Legyen $\sharp$ egy olyan függvény, melyre $\sharp : \Lambda \to \mathbb{N}$. Ilyen függvény könnyen előállítható, például:

1. ha $x_1, x_2, \ldots$ a λ-kalkulus változói, akkor legyen $\sharp x_i = 2^i$,
2. a $\lambda x_i.E$ λ-absztrakcióra legyen $\sharp(\lambda x_i.E) = 3^{\sharp x_i}5^{\sharp E}$,
3. az E_1E_2 applikációra legyen $\sharp(E_1E_2) = 7^{\sharp E_1}11^{\sharp E_2}$.

1.5.6. Definíció. Gödel-szám:

Ha E egy λ-kifejezés, akkor a $\sharp E$ természetes számot az E λ-kifejezés Gödel-számának nevezzük.

A $\sharp E$ Gödel-szám Church-számjegyét $\ulcorner\sharp E\urcorner$-vel jelöljük, erre gyakran az $\underline{\sharp E}$ és $\lceil E\rceil$ jelölést is használják. Bebizonyítható a következő állítás:

1.5.7. Tétel. (A 2. fixpont tétel)

Minden E λ-kifejezéshez létezik olyan F λ-kifejezés, amelyre $E\ulcorner\sharp F\urcorner = F$.

Formálisan a tétel az 1. fixpont tételtől csak abban különbözik, hogy itt a baloldalon nem az F, hanem az F Gödel-számának Church-számjegye áll.

Bizonyítás. Legyen *app* és *num* a természetes számokon értelmezett két olyan függvény, amelyre

$app(\sharp G_1, \sharp G_2) = \sharp(G_1G_2)$,

és $n \in \mathbb{N}$-re

$num(n) = \sharp\ulcorner n\urcorner$.

Az *app* és *num* függvények λ-kifejezéseit (lásd 1.6.10. tétel) jelöljük app és num-mal, azaz legyen

$\mathsf{app}\ulcorner\sharp G_1\urcorner\ulcorner\sharp G_2\urcorner = \ulcorner\sharp(G_1G_2)\urcorner$,
$\mathsf{num}\ulcorner n\urcorner = \ulcorner\sharp\ulcorner n\urcorner\urcorner$.

A második egyenlet speciálisan egy $n = \sharp G$ természetes számra

$\mathsf{num}\ulcorner\sharp G\urcorner = \ulcorner\sharp\ulcorner\sharp G\urcorner\urcorner$.

Most legyen $H \equiv \lambda x.E(\mathsf{app}\,x(\mathsf{num}\,x))$, és legyen $F \equiv H\ulcorner\sharp H\urcorner$. Erre az F-re teljesül a tétel állítása:

$F \equiv H\ulcorner\sharp H\urcorner \equiv$
$\lambda x.E(\mathsf{app}\,x(\mathsf{num}\,x))\ulcorner\sharp H\urcorner \twoheadrightarrow_\beta$
$E(\mathsf{app}\ulcorner\sharp H\urcorner(\mathsf{num}\ulcorner\sharp H\urcorner)) =$

$E(\mathsf{app}\ulcorner\sharp H\urcorner\ulcorner\sharp\ulcorner\sharp H\urcorner\urcorner) =$
$E(\ulcorner\sharp(H\ulcorner\sharp H\urcorner)\urcorner) =$
$E(\ulcorner\sharp F\urcorner).$ □

1.5.2. A rekurzív függvények

Most térjünk vissza a fac meghatározásához. Láttuk, hogy $\mathsf{fac} = \mathsf{H}\ \mathsf{fac}$, azaz a fac a H λ-kifejezés fixpontja, így $\mathsf{fac} \equiv \mathsf{Y}\ \mathsf{H}$ választással rekurzió nélkül leírhatjuk a fac függvényt.

1.5.8. Példa. *(A 2! értékének kiszámítása)*

$\mathsf{fac}\ 2 \equiv$
$\mathsf{Y}\ \mathsf{H}\ 2 \equiv$
$\underline{(\lambda x.(\lambda y.x(yy))((\lambda y.x(yy)))\mathsf{H}}\ 2 \twoheadrightarrow_\beta$
$\mathsf{H}((\lambda y.\mathsf{H}(yy))(\lambda y.\mathsf{H}(yy)))2 \equiv$
$\underline{(\lambda fn.\mathsf{if}(= n\ 0)1(*\ n(f(-\ n\ 1))))((\lambda y.\mathsf{H}(yy))(\lambda y.\mathsf{H}(yy)))}2 \twoheadrightarrow_\beta$
$\underline{(\lambda n.\mathsf{if}(= n\ 0)1(*\ n(((\lambda y.\mathsf{H}(yy))(\lambda y.\mathsf{H}(yy)))(-\ n\ 1))))2} \twoheadrightarrow_\beta$
$\mathsf{if}(= 2\ 0)1(*\ 2\ (((\lambda y.\mathsf{H}(yy))(\lambda y.\mathsf{H}(yy)))(-\ 2\ 1))) \twoheadrightarrow_\delta$
$*\ 2((\lambda y.\mathsf{H}(yy))(\lambda y.\mathsf{H}(yy))1).$

A szorzás művelet második argumentuma:

$\underline{(\lambda y.\mathsf{H}(yy))(\lambda y.\mathsf{H}(yy))}1 \twoheadrightarrow_\beta$
$\mathsf{H}((\lambda y.\mathsf{H}(yy))(\lambda y.\mathsf{H}(yy)))1 \equiv$
$\underline{(\lambda fn.\mathsf{if}(= n\ 0)1(*\ n(f(-\ n\ 1))))((\lambda y.\mathsf{H}(yy))(\lambda y.\mathsf{H}(yy)))}1 \twoheadrightarrow_\beta$
$\underline{(\lambda n.\mathsf{if}(= n\ 0)1(*\ n(((\lambda y.\mathsf{H}(yy))(\lambda y.\mathsf{H}(yy)))(-\ n\ 1))))1} \twoheadrightarrow_\beta$
$\mathsf{if}(= 1\ 0)1(*\ 1(((\lambda y.\mathsf{H}(yy))(\lambda y.\mathsf{H}(yy)))(-\ 1\ 1))) \twoheadrightarrow_\delta$
$*\ 1((\lambda y.\mathsf{H}(yy))(\lambda y.\mathsf{H}(yy))0).$

Az előbbihez hasonlóan:

$\underline{(\lambda y.\mathsf{H}(yy))(\lambda y.\mathsf{H}(yy))}0 \twoheadrightarrow_\beta$
$\mathsf{H}((\lambda y.\mathsf{H}(yy))(\lambda y.\mathsf{H}(yy)))0 \equiv$

$\underline{(\lambda fn.\mathsf{if}(= n\ 0)1(*\ n(f(-\ n\ 1))))((\lambda y.\mathsf{H}(yy))(\lambda y.\mathsf{H}(yy)))}0 \ \rightarrow_\beta$
$\underline{(\lambda n.\mathsf{if}(= n\ 0)1(*\ n(((\lambda y.\mathsf{H}(yy))(\lambda y.\mathsf{H}(yy)))(-\ n\ 1))))0} \ \rightarrow_\beta$
$\mathsf{if}(= 0\ 0)1(*\ 0(((\lambda y.\mathsf{H}(yy))(\lambda y.\mathsf{H}(yy)))(-\ 0\ 1))) \ \rightarrow_\delta \ 1.$

Tehát

$*\ 2(*\ 1\ 1) \ \rightarrow_\delta \ *\ 2\ 1 \ \rightarrow_\delta \ 2.$ □

A következő példában a $\mathsf{fac} \equiv \mathsf{Y\ H}$ helyett a $\mathsf{fac} = \mathsf{H\ fac}$ alakot használjuk.

1.5.9. Példa. *(A 2! értékének kiszámítása)*

$\mathsf{fac}\ 2 \equiv$
$\mathsf{H\ fac}\ 2 \equiv$
$\underline{\lambda f.(\lambda n.\mathsf{if}(= n\ 0)1(*\ n(f(-\ n\ 1))))\mathsf{fac}}\ 2 \ \rightarrow_\beta$
$\underline{(\lambda n.\mathsf{if}(= n\ 0)1(*\ n(\mathsf{fac}(-\ n\ 1))))2} \ \rightarrow_\beta$
$\mathsf{if}(= 2\ 0)1(*\ 2(\mathsf{fac}(-\ 2\ 1))) \ \rightarrow_\delta$
$*\ 2(\mathsf{fac}(-\ 2\ 1)) \ \rightarrow_\delta$
$*\ 2(\mathsf{fac}\ 1).$

A szorzás második operandusa:

$\mathsf{fac}\ 1 \equiv$
$\mathsf{H\ fac}\ 1 \equiv$
$\underline{\lambda f.(\lambda n.\mathsf{if}(= n\ 0)1(*\ n(f(-\ n\ 1))))\ \mathsf{fac}}1 \ \rightarrow_\beta$
$\underline{(\lambda n.\mathsf{if}(= n\ 0)1(*\ n(\mathsf{fac}(-\ n\ 1))))1} \ \rightarrow_\beta$
$\mathsf{if}(= 1\ 0)1(*\ 1(\mathsf{fac}(-\ 1\ 1))) \ \rightarrow_\delta$
$*\ 1(\mathsf{fac}(-\ 1\ 1)) \ \rightarrow_\delta$
$*\ 1(\mathsf{fac}\ 0).$

Határozzuk meg ennek a szorzásnak is a második operandusát:

$\mathsf{fac}\ 0 \equiv$
$\mathsf{H\ fac}\ 0 \equiv$

$\underline{\lambda f.(\lambda n.\mathsf{if}(= n\ 0)1(*\ n(f(-\ n\ 1))))\mathsf{fac}}\ 0 \twoheadrightarrow_\beta$
$\underline{(\lambda n.\mathsf{if}(= n\ 0)1(*\ n(\mathsf{fac}(-\ n\ 1))))0} \twoheadrightarrow_\beta$
$\mathsf{if}(= 0\ 0)1(*\ 0(\mathsf{fac}(-\ 0\ 1))) \twoheadrightarrow_\delta 1.$

Tehát

$*\ 2(*\ 1\ 1) \twoheadrightarrow_\delta *\ 2\ 1 \twoheadrightarrow_\delta 2.$ □

A fixpont-kombinátorokkal való számításokat leegyszerűsíti a következő észrevétel.

1.5.10. Állítás. (Fixpont-kombinátorok és a redukció)

$$\begin{array}{lll} \mathsf{Y}(\lambda uv.E)F & = & E[u := \mathsf{Y}(\lambda uv.E)][v := F], \\ \Theta(\lambda uv.E)F & \twoheadrightarrow_\beta^+ & E[u := \Theta(\lambda uv.E)][v := F]. \end{array}$$

Bizonyítás.
$\mathsf{Y}(\lambda uv.E)F \equiv$
$\underline{(\lambda x.(\lambda y.x(yy))(\lambda y.x(yy)))(\lambda uv.E)}F \twoheadrightarrow_\beta$
$\underline{(\lambda y.(\lambda uv.E)(yy))(\lambda y.(\lambda uv.E)(yy))}F \twoheadrightarrow_\beta$
$(\lambda uv.E)(\underline{(\lambda y.(\lambda uv.E)(yy))(\lambda y.(\lambda uv.E)(yy))})F \twoheadleftarrow_\beta$
$(\lambda uv.E)((\lambda x.(\lambda y.x(yy))(\lambda y.x(yy)))(\lambda uv.E))F \equiv$
$(\lambda uv.E)(\mathsf{Y}(\lambda uv.E))F \twoheadrightarrow_\beta^+$
$E[u := \mathsf{Y}(\lambda uv.E)][v := F].$

Az állítás második része is hasonlóan könnyen belátható, a levezetésben β-absztrakcióra nincs szükség. □

1.5.11. Példa. *(A 2! értékének kiszámítása)*
A 1.5.8. példa számításai a fenti állítás alkalmazásával jelentősen leegyszerűsödnek.

$\mathsf{fac}\ 2 \equiv$
$\mathsf{Y}\ \mathsf{H}\ 2 \equiv$
$\mathsf{Y}(\lambda fn.\mathsf{if}(= n\ 0)1(*\ n(f(-\ n\ 1))))2 =$
$\mathsf{if}(= 2\ 0)1(*\ 2\ ((\mathsf{Y}\ \mathsf{H})(-\ 2\ 1))) \twoheadrightarrow_\delta$

$* \; 2 \; ((\mathsf{Y} \; \mathsf{H})(- \; 2 \; 1)) \; \twoheadrightarrow_\delta$
$* \; 2 \; (\mathsf{Y} \; \mathsf{H} \; 1).$

A szorzás művelet második argumentuma:

$\mathsf{Y} \; \mathsf{H} \; 1 \equiv$
$\mathsf{Y}(\lambda fn.\mathsf{if}(= n \; 0)1(* \; n(f(- \; n \; 1))))1 =$
$\mathsf{if}(= 1 \; 0)1(* \; 1 \; ((\mathsf{Y} \; \mathsf{H})(- \; 1 \; 1))) \; \twoheadrightarrow_\delta$
$* \; 1 \; ((\mathsf{Y} \; \mathsf{H})(- \; 1 \; 1)) \; \twoheadrightarrow_\delta$
$* \; 1 \; (\mathsf{Y} \; \mathsf{H} \; 0).$

Az előbbihez hasonlóan:

$\mathsf{Y} \; \mathsf{H} \; 0 \equiv$
$\mathsf{Y}(\lambda fn.\mathsf{if}(= n \; 0)1(* \; n(f(- \; n \; 1))))0 =$
$\mathsf{if}(= 0 \; 0)1(* \; 0 \; ((\mathsf{Y} \; \mathsf{H})(- \; 0 \; 1))) \; \twoheadrightarrow_\delta$
$1.$ □

Összefoglalva, a módszer tehát a következő. Ha adott egy

$f = \ldots f \ldots$

rekurzív matematikai függvény, akkor először ebből készítünk egy

$\mathsf{F} \equiv \lambda x. \; \ldots x \ldots$

λ-kifejezést. Ha f-fel jelöljük az f függvény λ-kifejezését, akkor $\mathsf{f} = \mathsf{F} \; \mathsf{f}$, azaz az f függvény λ-kifejezését az F fixpontja, például az $\mathsf{Y} \; \mathsf{F}$ applikáció adja. Az $\mathsf{Y} \; \mathsf{F} \; E$ applikáció redukálásakor az 1.5.10. állítást használhatjuk.

1.5.3. A Russell-paradoxon

A naív halmazelmélet egyik ellentmondása a *Russell-paradoxon*: Nevezzünk egy halmazt „rendes” halmaznak, ha önmagát nem tartalmazza elemként, és legyen A az összes rendes halmazok halmaza, azaz $A = \{x \mid x \notin x\}$. Ekkor

- ha A egy rendes halmaz, akkor benne van A-ban, hiszen az A az összes rendes halmazt tartalmazza. Tehát A benne van A-ban, és így nem lehet rendes halmaz,
- ha A nem rendes halmaz, akkor önmagát tartalmazza, mert ellenkező esetben rendes halmaz lenne. Tehát A benne van A-ban, de mivel A a rendes halmazokat tartalmazza, A csak rendes lehet,

azaz $A \in A \Leftrightarrow A \notin A$. A naív halmazelmélet tehát *inkonzisztens*. Most azt mutatjuk meg, hogy ez az inkonzisztencia a λ-kalkulussal leírható.

Ha az $x \in B$ tartalmazásnak egy olyan $\mathsf{B}x$ applikációt feleltetünk meg, amelyre

$$\mathsf{B}\ x = \begin{cases} \mathsf{true}, & \text{ha } x \in B, \\ \mathsf{false}, & \text{ha } x \notin B, \end{cases}$$

és ha az $\{x \mid C\}$ halmazhoz a $\lambda x.\mathsf{C}$ λ-absztrakciót rendeljük, akkor a Russell-paradoxon A halmaza az

$$\mathsf{A} \equiv \lambda x.\mathsf{not}(xx)$$

λ-kifejezéssel írható le. Az $A \in A$ λ-kifejezése

$$\mathsf{A}\ \mathsf{A} \equiv (\lambda x.\mathsf{not}(xx))\mathsf{A} \ \twoheadrightarrow_\beta \ \mathsf{not}(\mathsf{A}\ \mathsf{A}),$$

ami nyilván ellentmondás. Ha tehát a λ-kifejezésekhez matematikai fogalmakat rendelünk, például a tartalmazást, a halmazt és a not függvényt, akkor ezekkel a matematikai fogalmakkal könnyen ellentmondásra juthatunk, azaz a bevezetett matematikai fogalmakkal, az általános értelemben vett függvényfogalommal a λ-kalkulus inkonzisztenssé válik. Church ezt a problémát úgy próbálta elkerülni, hogy azt mondta, az $\mathsf{A}\ \mathsf{A}$-nak „nincs értelme”, az $\mathsf{A} \equiv \lambda x.\mathsf{not}(xx)$ függvény ne legyen értelmezve az A pontban, azaz $\lambda x.\mathsf{not}(xx)$-ben. Ebből azonnal adódott, hogy a „nincs értelmezve” fogalomnak a „nincs normál formája” fogalmat kell megfeleltetni; ezzel a kérdéssel a következő pontban foglalkozunk.

Visszatérve a fenti egyenletre, az egyenletből azonnal látható, hogy az $\mathsf{A}\ \mathsf{A}$ a not fixpontja. Így

$\mathsf{Y}\ \mathsf{not} = \mathsf{A}\ \mathsf{A} \equiv (\lambda x.\mathsf{not}(xx))(\lambda x.\mathsf{not}(xx))$.

Ezt általánosítva, ha a not-ot egy tetszőleges f-fel helyettesítjük,

$\mathsf{Y}\ \mathsf{f} = (\lambda x.\mathsf{f}(xx))(\lambda x.\mathsf{f}(xx))$,

és ebből

$\mathsf{Y} \equiv \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$.

Az Y fixpont-kombinátor tehát a Russell-paradoxonból levezethető, és ezért nevezte Curry az Y-t paradoxon-kombinátornak.

1.6. λ-definiálható függvények

Az előző pontban láttuk, hogy a λ-kalkulusban a rekurzív függvények a fixpont kombinátor segítségével leírhatók. Most arra a kérdésre keresünk választ, hogy vajon minden függvény leírható-e λ-kalkulusban, és hogy a λ-kalkulusban leírható függvények rekurzív függvények-e.

A továbbiakban függvényen az $f : \mathbb{N}^n \to \mathbb{N}$ leképezést értjük. Először a teljes rekurzív függvényekkel, majd ezután a parciális rekurzív függvényekkel foglalkozunk. Rekurzív függvényen, ha külön nem jelöljük, mindig teljes rekurzív függvényt értünk. A természetes számok leírására a Church-számjegyrendszer λ-kifejezéseit használjuk.

1.6.1. Definíció. λ-definiálható függvény:

Azt mondjuk, hogy az f n-változós függvény λ-definiálható, ha ezt a függvényt az $F \in \Lambda^0$ λ-kifejezés reprezentálja, azaz minden $x_1, x_2, \ldots x_n$-re

$F\ulcorner x_1\urcorner\ulcorner x_2\urcorner\ldots\ulcorner x_n\urcorner = \ulcorner f(x_1, x_2, \ldots, x_n)\urcorner$.

Mivel a Church-számjegyrendszer normált számjegyrendszer, azaz minden számnak van normál formája, feltehetjük, hogy a definícióban szereplő számok normál formában vannak. Az I. Church-Rosser tétel alapján a fenti egyenlőség így is írható:

$F\ulcorner x_1\urcorner\ulcorner x_2\urcorner\ldots\ulcorner x_n\urcorner \twoheadrightarrow_\beta^+ \ulcorner f(x_1,x_2,\ldots,x_n)\urcorner$,

azaz a λ-kalkulusban a λ-definiálható függvények értékét normál sorrendű redukciókkal meghatározhatjuk.

1.6.1. Primitív rekurzív függvények

Most a primitív rekurzív függvényeket definiáljuk, és megmutatjuk, hogy a λ-kalkulusban a primitív rekurzív függvények leírhatók.

1.6.2. Definíció. Primitív rekurzív függvények:

Primitív rekurzív függvények a következő függvények:

1. $Z(x) = 0$, *a nulla értékű konstans függvény,*
2. $succ(x) = x + 1$, *az egységgel növelő függvény, és*
3. $U_i^n(x_1, x_2, \ldots, x_n) = x_i \quad (1 \le i \le n)$, *a szelektor függvény.*

Primitív rekurzív függvényekből további új primitív rekurzív függvényeket a helyettesítés és a primitív rekurzió véges sokszori alkalmazásával konstruálhatunk.

4. *Helyettesítés: legyen g egy m-változós primitív rekurzív függvény, és legyenek $h_1, h_2, \ldots, h_m$ n-változós primitív rekurzív függvények. Ekkor helyettesítéssel a következő primitív rekurzív f függvényt kapjuk:*
$f(x_1, x_2, \ldots, x_n) =$
$g(h_1(x_1, x_2, \ldots, x_n), h_2(x_1, x_2, \ldots, x_n), \ldots, h_m(x_1, x_2, \ldots, x_n))$.
5. *Primitív rekurzió: legyen g egy $n-1$-változós, és h egy $n+1$-változós primitív rekurzív függvény. A primitív rekurzóval a következő f primitív rekurzív függvényt kapjuk:*
$$\begin{aligned} f(0, x_2, \ldots, x_n) &= g(x_2, \ldots, x_n) \\ f(succ(x_1), x_2, \ldots, x_n) &= h(f(x_1, x_2, \ldots, x_n), x_1, x_2, \ldots, x_n). \end{aligned}$$

A fenti definíció 1–3. pontjaiban leírt függvényeket *alapfüggvényeknek*, a primitív rekurzióban szereplő h függvényt pedig a rekurzió *lépésfüggvényének* nevezzük. Bebizonyítható, hogy a primitív rekurzió minden adott

g és h függvényhez pontosan egy függvényt határoz meg, ezt az állítást a *primitív rekurzió tételének* nevezzük. A primitív rekurzív függvények jelentőségét az adja meg, hogy minden primitív rekurzív függvény *Turing-kiszámítható*, minden primitív rekurzív függvényhez létezik olyan Turing gép, amellyel a számítás véges sok lépésben véget ér.

Megjegyezzük, hogy a primitív rekurzió f függvényének második sorát gyakran az

$$f(succ(x_1), x_2, \ldots, x_n) = h(x_1, f(x_1, x_2, \ldots, x_n), x_2, \ldots, x_n).$$

alakban adják meg.

1.6.3. Példa. *(Az $Id(x) = x$ identitás függvény primitív rekurzív)*

$$\begin{aligned} Id(0) &= Z(0) = 0, \\ Id(succ(x_1)) &= succ(U_1^2(Id(x_1), x_1)), \end{aligned}$$

ahol a definícióban szereplő $h(x_1, x_2)$ függvényt az $U_1^2(x_1, x_2)$-nak a $succ(x)$-be helyettesítésével kaptuk meg. □

1.6.4. Példa. *(A konstans függvények primitív rekurzívak)*
A $K_m(x) = m$ $(m \geq 0)$ konstans függvény:

$$\begin{aligned} K_m(0) &= Id(m) = m, \\ K_m(succ(x)) &= U_1^2(K_m(x), x). \end{aligned}$$ □

1.6.5. Példa. *(Az $sg(x)$ előjelfüggvény primitív rekurzív)*

$$\begin{aligned} sg(0) &= Z(0) = 0, \\ sg(succ(x)) &= K_1(U_1^2(sg(x), x)). \end{aligned}$$ □

1.6.6. Példa. *(Az $add(x_1, x_2) = x_1 + x_2$ függvény primitív rekurzív)*

$$\begin{aligned} add(0, x_2) &= Id(x_2) = x_2, \\ add(succ(x_1), x_2) &= succ(U_1^3(add(x_1, x_2), x_1, x_2)) = succ(add(x_1, x_2)). \end{aligned}$$ □

1.6.7. Példa. *(A $mul(x_1, x_2) = x_1 * x_2$ függvény primitív rekurzív)*
Mivel az $add(x_1, x_2)$ függvény primitív rekurzív,

$$\begin{aligned}
mul(0, x_2) &= Z(x_2) = 0,\\
mul(succ(x_1), x_2) &= add(U_1^3(mul(x_1, x_2), x_1, x_2), U_2^3(mul(x_1, x_2), x_1, x_2)) =\\
&\quad add(mul(x_1, x_2), x_2),
\end{aligned}$$

ahol a jobboldalon szereplő függvényt a helyettesítés kétszeri alkalmazásával kaptuk meg. □

Most megmutatjuk, hogy a fenti definíció 1–5. pontjában megadott függvények a λ-kalkulusban leírhatók, a függvények λ-definiálhatósága nyilvánvaló. A fenti definíció első két pontjában szereplő függvények λ-kifejezéseit már ismerjük, a szelektor függvény λ-kifejezése is egyszerű:

i) $\lambda n.\ulcorner 0 \urcorner \equiv \lambda nfx.x$,

ii) $\mathsf{succ} \equiv \lambda nfx.nf(fx)$, és

iii) $\lambda x_1 x_2 \dots x_n.x_i$.

iv) A helyettesítés leírásához tegyük fel, hogy a definícióban szereplő $g, h_1, h_2, \dots, h_m$ függvények λ-kifejezései rendre $G, H_1, H_2, \dots, H_m$. Ekkor az f függvény a következő F λ-kifejezéssel definiálható:

$$F \equiv$$
$$\lambda x_1 x_2 \dots x_n.G(H_1 x_1 x_2 \dots x_n)(H_2 x_1 x_2 \dots x_n) \dots (H_m x_1 x_2 \dots x_n).$$

v) A primitív rekurzió műveletét először írjuk át *if-then-else* alakba:

$$\begin{aligned}
f(x_1, x_2, \dots, x_n) \;=\; & \mathit{if}\ (x_1 = 0)\\
& \quad \mathit{then}\ g(x_2, \dots, x_n)\\
& \quad \mathit{else}\\
& h(f(pred(x_1), x_2, \dots, x_n), pred(x_1), x_2, \dots, x_n),
\end{aligned}$$

ahol a $pred(x)$ függvény a nem nulla x számra az x-et megelőző számot adja. Látható, hogy az f egy rekurzív függvény. Tegyük fel, hogy a definícióban szereplő g és h függvények λ-kifejezései G és H. A Church-számjegyekre már definiált if, zero és pred λ-kifejezéseket használva az f λ-kifejezése legyen

$$\begin{aligned} F \equiv \lambda x_1 x_2 \ldots x_n.\mathsf{if}\ &(\mathsf{zero}\ x_1)\\ &(G x_2 \ldots x_n)\\ &(H(F((\mathsf{pred}\ x_1)x_2 \ldots x_n)(\mathsf{pred}\ x_1)x_2 \ldots x_n)). \end{aligned}$$

Ebből

$$\begin{aligned} E \equiv \lambda y x_1 x_2 \ldots x_n.\mathsf{if}\ &(\mathsf{zero}\ x_1)\\ &(G x_2 \ldots x_n)\\ &(H(y((\mathsf{pred}\ x_1)x_2 \ldots x_n)(\mathsf{pred}\ x_1)x_2 \ldots x_n)), \end{aligned}$$

amelyre $F = EF$, és így az Y fixpont-kombinátor alkalmazásával $F = \mathsf{Y}E$, azaz

$$\begin{aligned} F \equiv \mathsf{Y}(\lambda y x_1 x_2 \ldots x_n.\mathsf{if}\ &(\mathsf{zero}\ x_1)\\ &(G x_2 \ldots x_n)\\ &(H((y(\mathsf{pred}\ x_1)x_2 \ldots x_n)(\mathsf{pred}\ x_1)x_2 \ldots x_n))). \end{aligned}$$

1.6.2. Teljes rekurzív függvények

Ha a primitív rekurzív függvények 1.6.2. definícióját kiegészítjük a *minimalizálás*, vagy más elnevezéssel, az *inverzió* művelettel, akkor a *teljes rekurzív függvényeket* kapjuk meg. Megmutatjuk, hogy a teljes rekurzív függvények is λ-definiálhatók.

1.6.8. Definíció. Teljes rekurzív függvények:

Legyenek a teljes rekurzív függvények a 1.6.2. definícióban szereplő 0 konstans, a succ és az U_i^n függvények. Teljes rekurzív függvényekből további új teljes rekurzív függvényeket a helyettesítés, primitív rekurzió és a minimalizálás véges sokszori alkalmazásával konstruálhatunk. A helyettesítést és a primitív rekurziót az 1.6.2.-ban definiáltuk, a minimalizálás a következő:

6. *Minimalizálás (vagy inverzió): Legyen* g *egy* $n+1$ *-változós teljes rekurzív függvény, és tegyük fel, hogy minden adott* $x_1, x_2, \dots, x_n$ *-re mindig létezik olyan* y*, amelyre* $g(y, x_1, x_2, \dots, x_n) = 0$*. Jelölje*

   $$\mu y[g(y, x_1, x_2, \dots, x_n) = 0]$$

   a legkisebb olyan y *számot, amelyre* $g(y, x_1, x_2, \dots, x_n) = 0$*. Ekkor* g *-ből minimalizálással a következő teljes rekurzív függvényt kapjuk:*

   $$f(x_1, x_2, \dots, x_n) = \mu y[g(y, x_1, x_2, \dots, x_n) = 0].$$

A fenti definícióban a „teljes" jelző azt jelenti, hogy minden adott $x_1, x_2, \dots, x_n$ -re mindig létezik olyan y, amely kielégíti a

$$g(y, x_1, x_2, \dots, x_n) = 0$$

egyenletet. A μ-operátor a legkisebb ilyen y -t határozza meg. Minden teljes rekurzív függvény *Turing-kiszámítható*, minden teljes rekurzív függvényhez létezik olyan Turing gép, amellyel a számítás véges sok lépésben véget ér.

Most megmutatjuk, hogy a minimalizálás függvénye is leírható λ-kalkulusban.

vi) Az 1.6.8. definíció 6. pontjában leírt egyenlet jobboldalát adott g függvényre és adott $x_1, x_2, \dots, x_n$ értékekre állítsa elő egy

$$muop_{g,x_1,x_2,\dots,x_n}(z) \equiv muop(z)$$

függvény úgy, hogy g -t sorban tesztelje a $z = 0, 1, \dots$ értékekre, és a legkisebb olyan z érték legyen az y, amelyre a $g(z, x_1, x_2, \dots, x_n) = 0$ állítás teljesül. A *muop* függvényt *if-then-else* alakban felírva:

$$\begin{aligned} muop(z) \;&=\; \mathit{if}\; g(z, x_1, x_2, \dots, x_n) = 0 \\ & \mathit{then}\; z \\ & \mathit{else}\; muop(succ(z)), \end{aligned}$$

ahol a z -nek, azaz a *muop* argumentumának a kezdőértéke nulla. A fenti egyenletet λ-kifejezésre átírva, ha a *muop* λ-kifejezése muop és a g teljes rekurzív függvény λ-kifejezése G,

$$\begin{aligned}\mathsf{muop} \;\equiv \lambda z.\mathsf{if}\ & (\mathsf{zero}(G z\, \mathsf{x_1}\, \mathsf{x_2} \dots \mathsf{x_n}))\\ & z\\ & (\mathsf{muop}(\mathsf{succ}\, z)).\end{aligned}$$

Látható, hogy a **muop** függvény rekurzív, ebből pedig a már többször alkalmazott eljárással, az Y fixpont kombinátor felhasználásával, megkaphatjuk a **muop** rekurziómentes leírását:

$$\begin{aligned}\mathsf{muop} \;\equiv \mathsf{Y}(\lambda yz.\mathsf{if}\ & (\mathsf{zero}(G z\, \mathsf{x_1}\, \mathsf{x_2} \dots \mathsf{x_n}))\\ & z\\ & (y(\mathsf{succ}\, z))).\end{aligned}$$

A **muop** függvény a nulla kezdőértékre adja a minimális z értéket, így a g-ből a minimalizálással konstruálható f teljes rekurzív függvény λ-kifejezése

$$\begin{aligned}F \equiv \lambda x_1 x_2 \dots x_n.\mathsf{Y}(\lambda yz.\mathsf{if}\ & (\mathsf{zero}(G z x_1 x_2 \dots x_n))\\ & z\\ & (y(\mathsf{succ}\, z)))\ulcorner 0 \urcorner.\end{aligned}$$

1.6.9. Példa. *(Az F függvény helyes működése könnyen igazolható.)* Tegyük fel, hogy $n = 2$, egy adott g háromváltozós teljes rekurzív függvényre és c_1, c_2 konstansokra $\mu y[g(y, c_1, c_2) = 0] = 2$.

Ha E-vel jelöljük a $\lambda yz.\mathsf{if}(\mathsf{zero}(G z\, \mathsf{c_1}\, \mathsf{c_2}))z(y(\mathsf{succ}\, z))$ kifejezést,

$F\, \mathsf{c_1}\, \mathsf{c_2} \;\twoheadrightarrow_\beta^+$
$\mathsf{Y}(\lambda yz.\mathsf{if}(\mathsf{zero}(G z\, \mathsf{c_1}\, \mathsf{c_2}))z(y(\mathsf{succ}\, z)))\ulcorner 0 \urcorner \equiv$
$\mathsf{Y}E \ulcorner 0 \urcorner.$

Ez a kifejezés az 1.5.10. állítás felhasználásával tovább redukálható,

$\twoheadrightarrow_\beta^+$
$(\mathsf{if}(\mathsf{zero}(G z\, \mathsf{c_1}\, \mathsf{c_2}))z(y(\mathsf{succ}\, z)))[y := \mathsf{Y}E][z := \ulcorner 0 \urcorner] \equiv$
$\mathsf{if}(\mathsf{zero}(G \ulcorner 0 \urcorner \mathsf{c_1}\, \mathsf{c_2}))\ulcorner 0 \urcorner(\mathsf{Y}E(\mathsf{succ} \ulcorner 0 \urcorner)) \;\twoheadrightarrow_\beta^+$
$\mathsf{Y}E(\mathsf{succ}\, \ulcorner 0 \urcorner) \;\twoheadrightarrow_\beta$

$\mathsf{Y}E\ulcorner 1\urcorner$.

Megismételve ezeket a lépéseket az $\ulcorner 1\urcorner$ argumentumra,

$\twoheadrightarrow_\beta^+$
$(\mathsf{if}(\mathsf{zero}(Gz\,\mathsf{c}_1\,\mathsf{c}_2))z(y(\mathsf{succ}\,x)))[y := \mathsf{Y}E][z := \ulcorner 1\urcorner] \equiv$
$\mathsf{if}(\mathsf{zero}(G\ulcorner 1\urcorner\,\mathsf{c}_1\,\mathsf{c}_2))\ulcorner 1\urcorner(\mathsf{Y}E(\mathsf{succ}\ulcorner 1\urcorner)) \twoheadrightarrow_\beta^+$
$\mathsf{Y}E(\mathsf{succ}\ulcorner 1\urcorner) \twoheadrightarrow_\beta$
$\mathsf{Y}E\ulcorner 2\urcorner$.

Ebből

$\twoheadrightarrow_\beta^+$
$(\mathsf{if}(\mathsf{zero}(Gz\,\mathsf{c}_1\,\mathsf{c}_2))z(y(\mathsf{succ}\,x)))[y := \mathsf{Y}E][z := \ulcorner 2\urcorner] \equiv$
$\mathsf{if}(\mathsf{zero}(G\ulcorner 2\urcorner\,\mathsf{c}_1\,\mathsf{c}_2))\ulcorner 2\urcorner(\mathsf{Y}E(\mathsf{succ}\ulcorner 2\urcorner)) \twoheadrightarrow_\beta^+$
$\ulcorner 2\urcorner$,

így

$F\,\mathsf{c}_1\,\mathsf{c}_2 \twoheadrightarrow_\beta^+ \ulcorner 2\urcorner$. □

Tehát az *i–vi.* pontban leírt függvényekkel és konstrukciókkal a teljes rekurzív függvények λ-definiálhatók. Az állítás fordítva is igaz, azaz az 1.6.1. definícióval λ-definiálható minden függvény teljes rekurzív, így a következő tételt kapjuk:

1.6.10. Tétel. (Kleene-tétel [1936.])

A λ-definiálható függvények osztálya pontosan azonos a teljes rekurzív függvények osztályával.

1.6.3. Parciális rekurzív függvények

Egy f parciális rekurzív függvény értéke nincs minden $x_1, x_2, \ldots, x_n$ argumentumra definiálva. Ha ennek az f függvénynek akarjuk a λ-kifejezését meghatározni, akkor az a kérdés, hogy az f-hez milyen λ-kifejezéseket kell hozzárendelni azokban a pontokban, amelyekben az értéke nincs értelmezve. A klasszikus megoldást Church javasolta: ha az $f(x_1, x_2, \dots x_n)$ nincs értelmezve, akkor az $F\ulcorner x_1\urcorner\ulcorner x_2\urcorner\dots\ulcorner x_n\urcorner$ egy olyan λ-kifejezés legyen, amelynek nincs normál formája.

1.6.11. Definíció. λ-definiálható parciális függvény:

Legyen f egy n-változós parciális függvény, és reprezentálja ezt a függvényt az $F \in \Lambda^0$ λ-kifejezés. Azt mondjuk, hogy az f függvény λ-definiálható, ha minden $x_1, x_2, \dots x_n$-re

$$F\ulcorner x_1\urcorner\ulcorner x_2\urcorner\dots\ulcorner x_n\urcorner \begin{cases} = \ulcorner m\urcorner, & \textit{ha} \quad f(x_1, x_2, \dots, x_n) = m, \\ \textit{nincs normál formája}, & \\ & \textit{ha} \quad f(x_1, x_2, \dots, x_n) \textit{ nincs értelmezve.} \end{cases}$$

Ennek a definíciónak azonban több kellemetlen tulajdonsága van.

1. Mint majd a 2. fejezetben látni fogjuk, a λ-kifejezések a kombinátor logika kifejezéseibe konvertálhatók. A normál forma azonban nem invariáns tulajdonság a λ-kalkulus és a kombinátor logika között (lásd **??** pont). Ezért nem garantálható, hogy egy parciális rekurzív függvénynek megfeleltetett λ-kifejezés transzformációja az adott függvénynek a kombinátor logikában megfeleltetett kifejezését adja meg. Előfordulhat, hogy a kombinátor logika egy kifejezése már normál formában van, de a neki megfeleltethető λ-kifejezésnek még van redexe.

2. Ha egyenlőség relációt definiálunk két olyan λ-kifejezés közé, amelyeknek nincs normál formájuk, akkor a λ-kalkulus inkonzisztenssé válik (1.8.5. következmény).

3. Egy másik probléma a helyettesítéssel kapcsolatos: két függvény kompozíciójának λ-kifejezése nem feltétlenül a λ-kifejezések kompozíciója:

   Legyen $f(n) = 0$ és $g(n)$ olyan, hogy semmilyen n-re sincs értelmezve. Ekkor nyilván $f \circ g = f(g(n))$-nek sincs értéke semmilyen n-re. Ugyanakkor, ha az f λ-kifejezése $F \equiv (\lambda xy.x)\ulcorner 0\urcorner$, és a g λ-kifejezése $G \equiv (\lambda xy.x)\Omega$, akkor

$F \circ G \equiv \lambda x.F(G\ x) = \lambda x.\ulcorner 0 \urcorner$,

ez pedig biztosan nem az $f \circ g$ λ-kifejezése.

A problémákat az okozza, hogy a „nincs értelmezve” fogalomnak a „nincs normál formája” van megfeleltetve. Ezeknek a problémáknak a megszüntetésére Barendregt és Wadsworth 1971-ben bevezette a „megoldhatóság” fogalmát.

1.7. A megoldhatóság

A parciális függvények λ-definiálhatóságának definíciójában a „nincs értelmezve” esetnek egy „nem megoldható” kifejezést fogunk majd megfeleltetni.

1.7.1. Megoldható λ-kifejezés

Ebben a pontban a $\lambda x.x$ λ-kifejezésnek kiemelt szerepe van, ezért itt erre a λ-kifejezésre az 1.1.4. példában bevezetett I jelölést használjuk.

1.7.1. Definíció. Megoldható λ-kifejezés:

Az $E \in \Lambda^0$ λ-kifejezést megoldhatónak nevezzük, ha léteznek olyan $F_1, F_2, \ldots, F_n$ $(n \geq 0)$ λ-kifejezések, amelyekre

$EF_1F_2 \ldots F_n = \mathsf{I}.$

Egy tetszőleges $E \in \Lambda$ λ-kifejezés megoldható, ha az E lezárása, azaz $\lambda\vec{x}.E$ megoldható.

Könnyen belátható, hogy egy $E \in \Lambda$ λ-kifejezés megoldhatósága független a $\lambda\vec{x}.E$ lezárásban szereplő x_i változók sorrendjétől.

A „megoldható” elnevezés onnan származik, hogy ha E megoldható, akkor az $E \cdots = G$ egyenletnek tetszőleges G kifejezésre van „megoldása”:

1.7.2. Következmény.

Ha E megoldható, akkor tetszőleges G-hez léteznek olyan $F_1, F_2, \ldots, F_n$ λ-kifejezések, melyekre

$EF_1F_2 \ldots F_n = G.$

Bizonyítás. Az állítás triviális, hiszen ha E megoldható, akkor léteznek olyan $F_1, F_2, \ldots, F_{n-1}$ kifejezések, melyekre

$EF_1F_2 \ldots F_{n-1} = \mathsf{I}.$

Ekkor legyen $F_n \equiv G$,

így

$EF_1F_2 \ldots F_n = \mathsf{I}\, F_n \equiv \mathsf{I}G = G.$ □

1.7.3. Példa. *(Megoldható λ-kifejezések)*
A $\mathsf{K} \equiv \textsf{true} \equiv \lambda xy.x$ és a $\textsf{false} \equiv \lambda xy.y$ λ-kifejezések megoldhatók, hiszen tetszőleges F-re

$(\lambda xy.x)\mathsf{I}\, F = \mathsf{I},$
$(\lambda xy.y)F\,\mathsf{I} = \mathsf{I}.$

Megoldható a $\lambda xy.xy$ és az $\mathsf{S} \equiv \lambda xyz.xz(yz)$ kifejezés is.

$(\lambda xy.xy)\mathsf{I} = \mathsf{I},$
$(\lambda xyz.xz(yz))\mathsf{I}\,\mathsf{I}\,\mathsf{I} = \mathsf{I}.$ □

1.7.4. Példa. *(Az Y fixpont-kombinátor)*
Az Y fixpont-operátor is megoldható, felhasználva az $\mathsf{Y}F = F(\mathsf{Y}F)$ egyenletet,

$\mathsf{Y}(\lambda x.\mathsf{I}) = (\lambda x.\mathsf{I})(\mathsf{Y}(\lambda x.\mathsf{I})) = \mathsf{I}.$

Az Y fixpont-kombinátornak semmilyen argumentumra sincs normál formája, ezért az 1.6.1. definíció szerint az Y λ-kifejezés egy olyan numerikus függvénynek feleltethető meg, amelyik semmilyen pontban sincs értelmezve. És Y-ra, erre a nem-értelmezett függvényre applikálva a minden pontban értelmezett $\lambda y.\mathsf{I}$ függvényt, megkapjuk ennek a függvénynek a fixpontját, azaz egy λ-definiált, minden pontban értelmezett függvényt. (Ha a λ-definiálhatóságban a „nincs értelmezve” fogalomnak a „nem megoldható” fogalmat feleltetjük meg, akkor, mivel az Y megoldható, az Y „értelmezetté” válik, és így ez a furcsa tulajdonság megszűnik.) □

1.7.5. Példa. *(Az egyszerű számjegyrendszer számjegyei megoldhatók)*
Az $n \in \mathbb{N}$ szám egyszerű számjegyrendszerbeli λ-kifejezésére a $\mathsf{K\,I\,I}$ kifejezést applikáljuk, akkor eredményül az I λ-kifejezést kapjuk.

$\ulcorner 0 \urcorner \mathsf{K\,I\,I} \equiv$
$(\lambda x.x)\mathsf{K\,I\,I} \;\rightarrow_\beta$
$\mathsf{K\,I\,I} = \mathsf{I},$

és $n \geq 1$-re

$\ulcorner n \urcorner \mathsf{K\,I\,I} \equiv$
$\mathsf{succ}\ulcorner n-1 \urcorner \mathsf{K\,I\,I} \equiv$
$(\lambda yx.x\,\mathsf{false}\,y)\ulcorner n-1 \urcorner \mathsf{K\,I\,I} =$
$\mathsf{K}\,\mathsf{false}\ulcorner n-1 \urcorner \mathsf{I\,I} =$
$\mathsf{false}\,\mathsf{I\,I} \equiv$
$(\lambda xy.y)\mathsf{I\,I} = \mathsf{I}.$ □

1.7.6. Példa. *(A Scott-számjegyek megoldhatók)*
Az $n \in \mathbb{N}$ szám Scott-számjegye $\ulcorner n \urcorner$. Ha erre a λ-kifejezésre az $\mathsf{I}(\mathsf{K\,I})$ kifejezést applikáljuk, akkor eredményül az I λ-kifejezést kapjuk.

$\ulcorner 0 \urcorner \mathsf{I}(\mathsf{K\,I}) \equiv$
$(\lambda xy.x)\mathsf{I}(\mathsf{K\,I}) = \mathsf{I},$

és $n \geq 1$-re

$\ulcorner n \urcorner \mathsf{I}(\mathsf{K\,I}) \equiv$
$\mathsf{succ}\ulcorner n-1 \urcorner \mathsf{I}(\mathsf{K\,I}) \equiv$
$(\lambda zxy.yz)\ulcorner n-1 \urcorner \mathsf{I}(\mathsf{K\,I}) =$

$\mathsf{K}\,\mathsf{I}\ulcorner n-1\urcorner = \mathsf{I}.$ □

1.7.7. Példa. *(A Church-számjegyek megoldhatók)*
Az $n \in \mathbb{N}$ szám Church-számjegye $\ulcorner n\urcorner$. Ha erre a λ-kifejezésre az $\mathsf{I}\,\mathsf{I}$ kifejezést applikáljuk, akkor eredményül az I λ-kifejezést kapjuk.

$\ulcorner 0\urcorner\,\mathsf{I}\,\mathsf{I} \equiv$

$(\lambda fx.x)\mathsf{I}\,\mathsf{I} = \mathsf{I},$

és $n \geq 1$-re

$$\begin{aligned}
&\ulcorner n\urcorner\,\mathsf{I}\,\mathsf{I} \equiv\\
&\mathsf{succ}\ulcorner n-1\urcorner\,\mathsf{I}\,\mathsf{I} \equiv\\
&(\lambda nfx.f(nfx))\ulcorner n-1\urcorner\,\mathsf{I}\,\mathsf{I} =\\
&\mathsf{I}(\ulcorner n-1\urcorner\,\mathsf{I}\,\mathsf{I}) =\\
&\ulcorner n-1\urcorner\,\mathsf{I}\,\mathsf{I} =\\
&(\underbrace{\mathsf{I}\ldots(\mathsf{I}(\mathsf{I}}_{n-1}\,\mathsf{I}))\ldots) = \mathsf{I}.
\end{aligned}$$ □

1.7.8. Példa. *(Az Ω nem megoldható)*
Tegyük fel, hogy az $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$ kifejezés megoldható, azaz léteznek olyan $E_1, E_2, \ldots E_n$ kifejezések, melyekre $\Omega\, E_1E_2\ldots E_n = \mathsf{I}$. Mivel I normál formában van, az 1.3.10. következmény szerint

$$\Omega\, E_1E_2\ldots E_n \twoheadrightarrow_\beta^* \mathsf{I}.$$

Mivel csak

$$\Omega \twoheadrightarrow_\beta^* \Omega$$

lehetséges, ezért valamilyen F kifejezésre csak az

$$\Omega\, E_1E_2\ldots E_n \twoheadrightarrow_\beta^* \Omega\, F$$

lehetséges. Az $\Omega\, F \neq \mathsf{I}$ semmilyen F-re sem, így

$$\Omega\, E_1E_2\ldots E_n \not\twoheadrightarrow_\beta^* \mathsf{I}.$$ □

1.7.9. Példa. *(Az $x\,\mathsf{I}\,\Omega$ kifejezés)*
Az $x\,\mathsf{I}\,\Omega$ is megoldható. A λ-kifejezés lezárására

$(\lambda x.x\,\mathsf{I}\,\Omega)(\lambda xy.x) = \mathsf{I}.$ □

1.7.10. Példa. *(A $\lambda x.xx$ kifejezés is megoldható)*
Most megmutatjuk, hogy egy nem megoldható kifejezés részkifejezése megoldható is lehet. Az $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$ kifejezés $\lambda x.xx$ részkifejezésére

$(\lambda x.xx)\mathsf{I} = \mathsf{I},$

azaz a $\lambda x.xx$ kifejezés megoldható. □

Könnyen beláthatók a következő állítások.

1.7.11. Lemma. (Megoldhatóság és az absztrakció)
Az E λ-kifejezés akkor és csak akkor megoldható, ha $\lambda x.E$ is megoldható.

Nyilvánvaló, ha E nem megoldható, akkor a $\lambda x.E$ kifejezés is egy nem megoldható kifejezés.

1.7.12. Lemma. (Megoldhatóság és az applikáció)
Ha az EF λ-kifejezés megoldható, akkor az E is megoldható.

Természetesen, az állítás fordított irányban nem igaz, ha E megoldható, akkor egy F kifejezésre az EF nem feltétlenül megoldható kifejezés (1.7.10. példa).

1.7.13. Lemma. (A nem megoldhatóság)
Ha az E λ-kifejezés nem megoldható, akkor minden F-re az $E[x := F]$ és EF nem megoldható λ-kifejezések.

A megoldhatóság eldöntéséhez meg kell találni az 1.7.1. definícióban szereplő $F_1, F_2, \ldots, F_n$ λ-kifejezéseket. A megoldhatóság azonban szintaktikus tulajdonság, az $F_1, F_2, \ldots, F_n$ függvények nélkül, a λ-kifejezés szerkezetéből is eldönthető, hogy a λ-kifejezés megoldható-e. Ezzel foglalkozunk a következő pontban.

1.7.2. A fej normál forma

A λ-kifejezést *applikációs λ-kifejezésnek* nevezzük, ha EF alakú, és *absztrakciós λ-kifejezésnek*, ha $\lambda x.E$ alakú. A konstansos típusnélküli λ-kifejezések 1.4.23. definíciójából azonnal adódik, hogy egy λ-kifejezés vagy változó, vagy konstans, vagy applikációs λ-kifejezés, vagy egy absztrakciós λ-kifejezés. Könnyen belátható a következő két állítás:

1.7.14. Lemma. (Az applikációs λ-kifejezés)

Minden E applikációs λ-kifejezés

$$E \equiv F_1 F_2 \dots F_n \quad (n \geq 2)$$

alakú, ahol F_1 nem applikációs λ-kifejezés.

1.7.15. Lemma. (Az absztrakciós λ-kifejezés)

Minden E absztrakciós λ-kifejezés

$$E \equiv \lambda x_1 x_2 \dots x_n . F \quad (n \geq 1)$$

alakú, ahol F nem absztrakciós λ-kifejezés.

Ezekből az állításokból azonnal látható, hogy egy E λ-kifejezés vagy

1. $\lambda x_1 x_2 \dots x_n . x F_1 F_2 \dots F_m \quad (n, m \geq 0)$, vagy
2. $\lambda x_1 x_2 \dots x_n . (\lambda x . F_0) F_1 F_2 \dots F_m \quad (n \geq 0, m \geq 1)$

alakú, mivel

- ha E egy változó vagy konstans, akkor az 1. formában írható fel, ahol $n = m = 0$,
- ha E egy applikációs λ-kifejezés, azaz $E_1 E_2 \dots E_k$ formájú, akkor az E_1-től függően
  - ha E_1 egy változó vagy konstans, akkor az 1. formában írható fel $n = 0$-ra,
  - ha E_1 λ-absztrakció, akkor a 2. formában írható fel, szintén $n = 0$-ra,

- ha E absztrakciós λ-kifejezés, vagyis $\lambda x_1 x_2 \ldots x_n.F$ formájú, akkor az F-től függően
  - ha F egy változó vagy konstans, vagy egy olyan applikáció, amelynek első tagja egy változó vagy konstans, akkor az 1. alakban írható fel,
  - ha a törzsben levő applikáció első eleme egy absztrakció, akkor a 2. alakban írható fel.

A 2. formában $m \geq 1$ miatt a $(\lambda x.F_0)F_1$ részkifejezés biztosan redukálható, ezt a λ-kifejezés *fej redexének* nevezzük. Tehát a 2. formájú λ-kifejezésnek mindig van fej redexe. Az 1. típusú λ-kifejezésben is lehet fej redex, akkor, ha x egy olyan δ-függvény, amelynek m-nél nem több paramétere van. Ha x egy változó, számkonstans vagy logikai konstans, akkor redex csak az $F_1 F_2 \ldots F_m$-ben lehet.

Ha egy λ-kifejezésnek van fej redexe, akkor ez biztosan a legbaloldalibb redex, de a legbaloldalibb redex természetesen nem feltétlenül fej redex. Ha a λ-kifejezésnek nincs fej redexe, akkor a λ-kifejezés *fej normál formában* van.

1.7.16. Definíció. Fej normál forma:

Ha egy λ-kifejezés

$\lambda x_1 x_2 \ldots x_n.x F_1 F_2 \ldots F_m \quad (n, m \geq 0)$

alakú, és az $x F_1 F_2 \ldots F_p$ nem redex semmilyen $p \leq m$-re sem, akkor azt mondjuk, hogy a λ-kifejezés fej normál formában van.

A definícióban x-szel egy változót, vagy egy konstanst (speciálisan egy δ-függvényt) jelöltünk.

Azt mondjuk, hogy az E λ-kifejezésnek *van fej normál formája*, ha vagy fej normál formában van, vagy létezik olyan E' λ-kifejezés, amelyik fej normál formában van és $E = E'$. A fej normál forma x változója a λ-kifejezés *fej változója.*

1.7.17. Példa. *(Fej normál forma)*
A következő kifejezések fej normál formában vannak:

x,
2,
$+$,
xE,
$x\Omega$,
$\mathsf{I} \equiv \lambda x.x$,
$x(\mathsf{I}\,y)$,
$2\,E$,
$+\,2$,
$\lambda x.2$,
$\lambda x.yE$,
$\lambda x.2\,3$,
$\lambda x.+\,2$. □

1.7.18. Példa. *(Egy kifejezésnek több fej normál formája is lehet)*
A $\lambda x.\mathsf{I}\,x(\mathsf{I}\,\mathsf{I})$ kifejezés nincs fej normál formában, de

$\lambda x.\underline{\mathsf{I}\,x}(\mathsf{I}\,\mathsf{I}) \;\rightarrow_\beta$
$\lambda x.x(\mathsf{I}\,\mathsf{I})$

már fej normál forma, és egy újabb redukcióval,

$\lambda x.x(\underline{\mathsf{I}\,\mathsf{I}}) \;\rightarrow_\beta$
$\lambda x.x\,\mathsf{I}$,

az előzőtől különböző fej normál formát kapunk. □

1.7.19. Példa. *(A következő kifejezések nincsenek fej normál formában)*
A kifejezésekben aláhúzással a fej redexeket jelöljük.

$\underline{+\,2\,3}$,

$\underline{(\lambda x.y)2}$,
$\lambda x.\underline{(\lambda y.z)E}F$,
$\lambda x.\underline{+\,2\,3}\,4$,
$\Omega \equiv \underline{(\lambda x.xx)(\lambda x.xx)}$. □

1.7.20. Példa. *(Az* Y *kifejezés)*
Az Y fixpont-kombinátor nincs fej normál formában, de fej normál formára hozható:

$$\mathsf{Y} \equiv \lambda x.\underline{(\lambda y.x(yy))(\lambda y.x(yy))} \twoheadrightarrow_\beta$$
$$\lambda x.x((\lambda y.x(yy))(\lambda y.x(yy))).$$ □

1.7.21. Példa. *(A következő kifejezéseknek nincs fej normál formájuk)*
$\Omega \equiv \underline{(\lambda x.xx)(\lambda x.xx)} \twoheadrightarrow_\beta \Omega \twoheadrightarrow_\beta \ldots$
$\lambda x.\Omega \twoheadrightarrow_\beta \lambda x.\Omega \twoheadrightarrow_\beta \ldots$ □

A megoldhatóság és a fej normál forma közötti kapcsolatot a következő tétel adja meg:

1.7.22. Tétel. (Wadsworth-tétel [1971.])
Az E λ-kifejezés akkor és csak akkor megoldható, ha E-nek van fej normál formája.

A fej normál forma a normál forma általánosításának tekinthető, ha egy λ-kifejezés normál formában van, akkor nyilván fej normál formában is van, de ez fordítva nem érvényes: ha egy λ-kifejezés fej normál formában van, akkor nem feltétlenül van normál formában, sőt az is lehetséges, hogy nincs is normál formája. Erre a második esetre egy egyszerű példa az Y fixpont-kombinátor.

A fej normál forma már tartalmazza – ha létezik, – a normál forma struktúráját, ugyanis ha

$$\lambda x_1x_2\ldots x_n.xE_1E_2\ldots E_m \twoheadrightarrow_\beta^+ F,$$

akkor

$$F \equiv \lambda x_1x_2\ldots x_n.xF_1F_2\ldots F_m,$$

a két kifejezésben az absztrakciók változói és az applikációk száma nem változik meg, a törzs első eleme mindkét kifejezésben az x változó. A törzs elemeinek $E_i \twoheadrightarrow_{\beta}^{+} F_i$ $(1 \leq i \leq n)$ konverziói egymástól függetlenül, párhuzamosan is elvégezhetők.

Ez azt jelenti, hogy ha redukálásokkal elértünk a kifejezés fej normál formáig, akkor már mindent tudunk a λ-kifejezés szerkezetéről, további redukálások a λ-kifejezés felépítését már nem változtatják meg.

Könnyen beláthatók a következő állítások.

1.7.23. Lemma. (Fej normál forma és az absztrakció)

Az E λ-kifejezésnek akkor és csak akkor van fej normál formája, ha a $\lambda x.E$ kifejezésnek is van fej normál formája.

1.7.24. Lemma. (Fej normál forma és az applikáció)

Ha az EF λ-kifejezésnek van fej normál formája, akkor az E kifejezésnek is van fej normál formája.

1.7.25. Lemma. (Fej normál forma és a helyettesítés)

Ha az E λ-kifejezésnek van fej normál formája, akkor minden F-re az $E[x := F]$ kifejezésnek is van fej normál formája.

A lemmák szerint, ha az E λ-kifejezésnek nincs fej normál formája, akkor nincs az $EF_1F_2 \ldots F_n$ $(n \geq 1)$ kifejezésnek, nincs $\lambda x.E$-nek és az $E[x := F]$-nek sem. Az E úgy viselkedik, mint egy „teljesen definiálatlan" függvény. Ezért azok a λ-kifejezések, amelyeknek nincs fej normál formájuk, – egy kis túlzással – a λ-kifejezések „fekete lyukainak" tekinthetők.

1.7.3. A gyenge fej normál forma

Az 1. fejezet bevezetőjében utaltunk rá, hogy a funkcionális programok λ-kifejezések, az 1.1. pontban megmutattuk, hogy ezek csak zárt kifejezések lehetnek. Az 1.3. pontban foglalkoztunk ezeknek a kifejezéseknek az egyszerűsítésével, és láttuk, hogy az egyszerűsítés eredménye, azaz a program végrehajtásának eredménye a kifejezés normál formája.

A normál formát, ha létezik, a legbaloldalibb, legkülső redexek β-redukcióival, azaz a normál sorrendű redukálási stratégiával kapjuk meg. A

β-redukciónál azonban állandóan vizsgálni kell a névkonfliktust, azaz azt, hogy az argumentum szabad változója a redukálás után kötötté válik-e. Ha a névkonfliktus fennáll, akkor α-konverziót kell végrehajtani.

Mivel a programok kifejezései zárt kifejezések, ezért ha nem hajtunk végre a függvények belsejében redukciót, akkor nem fordulhat elő névkonfliktus. Ez azt jelenti, hogy a funkcionális programok kifejezéseinek redukálásaival megállhatunk, ha egy redukció eredménye függvény. Így jutunk el a *gyenge fej normál formához.*

1.7.26. Definíció. Gyenge fej normál forma:

Ha egy λ-kifejezés

$\lambda x.E$

vagy

$xF_1F_2\ldots F_n \quad (n \geq 0)$

alakú, és az $xF_1F_2\ldots F_m$ nem redex semmilyen $m \leq n$ -re sem, akkor azt mondjuk, hogy a λ-kifejezés gyenge fej normál formában van.

A definíció második kifejezésében x-szel egy változót, vagy egy konstanst (speciálisan egy δ-függvényt) jelöltünk.

Azt mondjuk, hogy az E λ-kifejezésnek *van gyenge fej normál formája*, ha vagy gyenge fej normál formában van, vagy létezik olyan E' λ-kifejezés, amelyik gyenge fej normál formában van és $E = E'$.

1.7.27. Példa. *(Gyenge fej normál forma)*
A következő kifejezések gyenge fej normál formában vannak:

x,

2,

$+$,

xE,

$\mathsf{I} \equiv \lambda x.x$,

$x(\mathsf{I}\,y)$,

$2\,E$,

$+\,2$,

$\lambda x.2$,

$\lambda x.yE$,

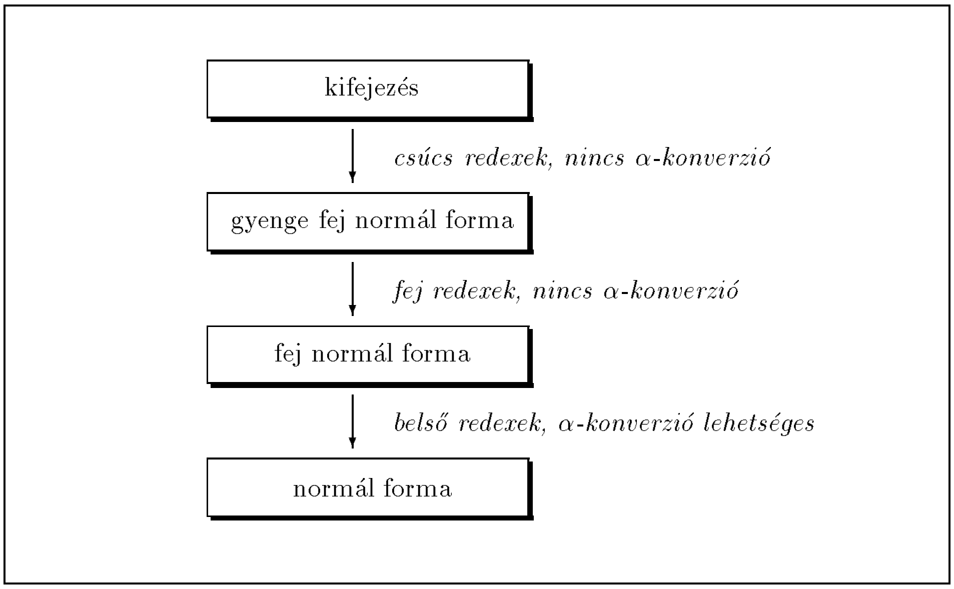
$\lambda x.2\,3$,

$\lambda x. +\ 2$,

$\lambda x.(\lambda y.E)F$ □

Nyilvánvaló, hogy ha egy kifejezés fej normál formában van, akkor gyenge fej normál formában is, de ez fordítva nem áll fenn. Az előző példa utolsó kifejezése gyenge fej normál formában van, de van egy redukálható fej redexe, azaz nincs fej normál formában.

Ha egy kifejezés nincs gyenge fej normál formában, akkor a legbaloldalibb legkülső redexet *csúcs redexnek* nevezzük. A gyenge fej normál formát a csúcs redexek redukálásaival kapjuk meg (1.15. ábra).



1.15. ábra. Redukálások és normál formák

1.7.28. Példa. *(Nem gyenge fej normál formák)*
A kifejezésekben aláhúzással a csúcs redexeket jelöljük.

$\underline{+\,2\,3}$,
$\underline{(\lambda x.y)2}$,
$\underline{(\lambda x.(\lambda y.z)x)x}z(\lambda y.z)$,
$\Omega \equiv \underline{(\lambda x.xx)(\lambda x.xx)}$. □

A fej normál formához hasonlóan, látható, hogy egy gyenge fej normál formában levő kifejezés struktúrája a további lehetséges redukálások végrehajtásával nem változik meg, a függvény függvény marad, az első változó vagy konstans sem fog a kifejezés elejéről eltűnni. Ezért a funkcionális programok végrehajtását, azaz a redukálásokat be lehet fejezni a gyenge fej normál forma elérésekor, azaz a funkcionális program eredménye mindig egy gyenge fej normál formájú kifejezés.

1.7.4. λ-definiálható függvények

Most definiáljuk újra a λ-definiálható függvényeket a fej normál forma felhasználásával.

1.7.29. Definíció. λ-definiálható parciális függvény:

Legyen f egy n-változós parciális függvény, és reprezentálja ezt a függvényt az $F \in \Lambda^0$ λ-kifejezés. Azt mondjuk, hogy az f függvény λ-definiálható, ha minden $x_1, x_2, \ldots x_n$-re

$$F\ulcorner x_1\urcorner\ulcorner x_2\urcorner\ldots\ulcorner x_n\urcorner \quad \begin{cases} = \ulcorner m\urcorner, \quad \textit{ha} \quad f(x_1, x_2, \ldots, x_n) = m, \\ \textit{nincs fej normál formája}, \\ \qquad\qquad \textit{ha} \quad f(x_1, x_2, \ldots, x_n) \ \textit{nincs értelmezve}. \end{cases}$$

A definíció és az 1.7.22. tétel alapján tehát a nem-értelmezett pontokban a parciális függvénynek egy olyan λ-kifejezést feleltetünk meg, amelyik az adott pontban, azaz az adott argumentumokra nem megoldható.

A parciális függvények λ-definiálhatóságának ezzel a definíciójával a korábban említett „kellemetlen” tulajdonságok megszűnnek, a λ-kalkulus és a kombinátor logika kifejezései konvertálhatók, és ha azokat a kifejezéseket, amelyeknek nincs fej normál formájuk, egyenlőeknek tekintjük, a λ-kalkulus konzisztens marad.

Most először a parciális rekurzív függvényeket definiáljuk, majd megmutatjuk, hogy a λ-kalkulusban a parciális rekurzív függvények leírhatók.

1.7.30. Definíció. Parciális rekurzív függvények:

Parciális rekurzív függvények a következő függvények:

1. $Z(x) = 0$, *a nulla értékű konstans függvény,*
2. $succ(x) = x + 1$, *az egységgel növelő függvény, és*
3. $U_i^n(x_1, x_2, \ldots, x_n) = x_i \quad (1 \leq i \leq n)$, *a szelektor függvény.*

Parciális rekurzív függvényekből további új parciális rekurzív függvényeket a helyettesítés, primitív rekurzió és a minimalizálás véges sokszori alkalmazásával konstruálhatunk.

4. *Helyettesítés: legyen* g *egy* m*-változós parciális rekurzív függvény, és legyenek* $h_1, h_2, \ldots, h_m$ n*-változós parciális rekurzív függvények. Ekkor helyettesítéssel a következő parciális rekurzív* f *függvényt kapjuk:*

   $$f(x_1, x_2, \ldots, x_n) =$$
   $$g(h_1(x_1, x_2, \ldots, x_n), h_2(x_1, x_2, \ldots, x_n), \ldots, h_m(x_1, x_2, \ldots, x_n)).$$

   Ha az $x_1, x_2, \ldots, x_n$ *pontokban a* $h_j(x_1, x_2, \ldots, x_n)$ $(1 \leq j \leq m)$ *függvények mindegyike és a*

   $$g(h_1(x_1, x_2, \ldots, x_n), h_2(x_1, x_2, \ldots, x_n), \ldots, h_m(x_1, x_2, \ldots, x_n))$$

   is értelmezett, akkor a helyettesítéssel kapott függvény is értelmezett, ellenkező esetben a függvény ezen a helyen nincs értelmezve.
5. *Primitív rekurzió: legyen* g *egy* $n-1$*-változós, és* h *egy* $n+1$*-változós parciális rekurzív függvény. A primitív rekurzóval a következő* f *parciális rekurzív függvényt kapjuk:*

$$f(0, x_2, \ldots, x_n) = g(x_2, \ldots, x_n)$$
$$f(succ(x_1), x_2, \ldots, x_n) = h(f(x_1, x_2, \ldots, x_n), x_1, x_2, \ldots, x_n).$$

Ha $g(x_2, \ldots, x_n)$ és $h(f(x_1, x_2, \ldots, x_n), x_1, x_2, \ldots, x_n)$ az adott pontokban értelmezett, akkor a primitív rekurzióval kapott függvény is értelmezett, ellenkező esetben a függvény ezen a helyen nincs értelmezve.

6. *Minimalizálás (vagy inverzió): Legyen g egy $n+1$-változós parciális rekurzív függvény. Ha adott $x_1, x_2, \ldots, x_n$-re mindig létezik olyan y, amelyre $g(y, x_1, x_2, \ldots, x_n) = 0$, és minden $(0 \leq z < y)$-ra $g(z, x_1, x_2, \ldots, x_n)$ értelmezett és $g(z, x_1, x_2, \ldots, x_n) \neq 0$, akkor jelölje ezt az y-t*

$$\mu y[g(y, x_1, x_2, \ldots, x_n) = 0],$$

és a g függvényből minimalizálással a következő parciális rekurzív függvényt kapjuk:

$$f(x_1, x_2, \ldots, x_n) = \mu y[g(y, x_1, x_2, \ldots, x_n) = 0].$$

Ellenkező esetben, azaz ha nincs ilyen y vagy egy z értékre a $g(z, x_1, x_2, \ldots, x_n)$ nincs értelmezve, akkor az $f(x_1, x_2, \ldots, x_n)$ függvény sincs az $x_1, x_2, \ldots, x_n$ pontban értelmezve.

A fenti definíció első három pontjában szereplő függvények λ-kifejezései azonosak a primitív rekurzív függvényeknél, az 1.6.1. pontban az *i.* – *iii.*-ben megadott függvényekkel. Most megadjuk a fenti definícióban szereplő helyettesítés λ-kifejezését.

vii) A helyettesítés leírásához a $g, h_1, h_2, \ldots, h_m$ függvények λ-kifejezései legyenek rendre $G, H_1, H_2, \ldots, H_m$. Ekkor az f függvény a következő F λ-kifejezéssel definiálható:

$$F \equiv$$
$$\lambda x_1 x_2 \ldots x_n.(H_1 x_1 x_2 \ldots x_n \mathsf{I}\mathsf{I})(H_2 x_1 x_2 \ldots x_n \mathsf{I}\mathsf{I}) \ldots (H_m x_1 x_2 \ldots x_n \mathsf{I}\mathsf{I})$$
$$(G(H_1 x_1 x_2 \ldots x_n)(H_2 x_1 x_2 \ldots x_n) \ldots (H_m x_1 x_2 \ldots x_n)).$$

Megmutatjuk, hogy azokban a pontokban, ahol az f függvény értelmezve van, az F λ-kifejezés előállítja az $f(x_1, x_2, \ldots, x_n)$ Church-számát. Ezután bebizonyítjuk azt, hogy ha valamelyik h_i nincs értelmezve, vagy ha a g függvény nincs értelmezve az adott pontban, akkor az $f(x_1, x_2, \ldots, x_n)$ λ-kifejezése nem megoldható.

Ha az $x_1, \ldots, x_n$ pontban $h_i(x_1, \ldots, x_n) \quad (1 \leq i \leq m)$ mindegyike értelmezett, akkor minden i-re

$$H_i \ulcorner x_1 \urcorner \ldots \ulcorner x_n \urcorner \mathsf{I}\mathsf{I} = \mathsf{I},$$

mivel $H_i \ulcorner x_1 \urcorner \ldots \ulcorner x_n \urcorner$ értéke egy Church-szám, és az 1.7.7. példában láttuk, hogy a Church-számok éppen az $\mathsf{I}\mathsf{I}$ kifejezéssel oldhatók meg. Ekkor tehát

$$\begin{aligned}
&F \ulcorner x_1 \urcorner \ldots \ulcorner x_n \urcorner \equiv \\
&\underbrace{\mathsf{I}\mathsf{I} \ldots \mathsf{I}}_{m} (G(H_1 \ulcorner x_1 \urcorner \ldots \ulcorner x_n \urcorner) \ldots (H_m \ulcorner x_1 \urcorner \ldots \ulcorner x_n \urcorner)) = \\
&G(H_1 \ulcorner x_1 \urcorner \ldots \ulcorner x_n \urcorner) \ldots (H_m \ulcorner x_1 \ldots \ulcorner x_n \urcorner) = \\
&G \ulcorner h_1(x_1, \ldots, x_n) \urcorner \ldots \ulcorner h_m(x_1, \ldots, x_n) \urcorner = \\
&\ulcorner g(h_1(x_1, \ldots, x_n), \ldots h_m(x_1, \ldots, x_n) \urcorner = \\
&\ulcorner f(x_1, \ldots, x_n) \urcorner.
\end{aligned}$$

Ha $h_i(x_1, x_2, \ldots, x_n)$ nincs definiálva valamelyik $1 \leq i \leq m$-re, akkor az $F \ulcorner x_1 \urcorner \ldots \ulcorner x_n \urcorner$ λ-kifejezés nem megoldható. Ha $h_{i_0}(x_1, \ldots, x_n)$ nincs értelmezve, akkor az 1.7.29. definíció szerint a

$$H_{i_0} \ulcorner x_1 \urcorner \ldots \ulcorner x_n \urcorner$$

kifejezés nem megoldható, így az 1.7.13. lemma szerint

$$H_{i_0} \ulcorner x_1 \urcorner \ldots \ulcorner x_n \urcorner \mathsf{I}\mathsf{I}$$

sem oldható meg, és így ugyanezen lemma alapján

$$F \ulcorner x_1 \urcorner \ldots \ulcorner x_n \urcorner$$

sem.

Ha mindegyik $h_i(x_1, \ldots, x_n) \quad (1 \leq i \leq m)$ definiált, és $h_i(x_1, \ldots, x_n) = c_i$, de $g(c_1, \ldots, c_m)$ nincs értelmezve, akkor

$$
\begin{aligned}
&F\ulcorner x_1 \urcorner \ldots \ulcorner x_n \urcorner \equiv \\
&\underbrace{\mathsf{I}\,\mathsf{I} \ldots \mathsf{I}}_{m}\,(G(H_1\ulcorner x_1 \urcorner \ldots \ulcorner x_n \urcorner) \ldots (H_m\ulcorner x_1 \urcorner \ldots \ulcorner x_n \urcorner)) = \\
&G(H_1\ulcorner x_1 \urcorner \ldots \ulcorner x_n \urcorner) \ldots (H_m\ulcorner x_1 \ldots \ulcorner x_n \urcorner) = \\
&G\ulcorner h_1(x_1, \ldots, x_n) \urcorner \ldots \ulcorner h_m(x_1, \ldots, x_n) \urcorner \equiv \\
&G\ulcorner c_1 \urcorner \ldots \ulcorner c_m \urcorner
\end{aligned}
$$

ez viszont, feltételünk szerint, az 1.7.29. definíció alapján nem megoldható.

Ezek után foglalkozzunk a primitív rekurzió és a minimalizálás λ-kifejezéseivel. A primitív rekurzió megadására nincs is szükség, mert Kleene *Normál Forma tétele* azt mondja ki, hogy minden f parciális rekurzív függvényhez léteznek olyan g és h primitív rekurzív függvények, hogy az f leírható e két függvénnyel, a helyettesítés és a minimalizálás műveleteinek felhasználásával.

viii) Ha az f és g függvények λ-kifejezései F és G, akkor a g-ből a minimalizálással előállított F függvény λ-kifejezése a következő:

$$
\begin{aligned}
F \equiv \lambda x_1 x_2 \ldots x_n.\mathsf{Y}(\lambda yz.\mathsf{if}\ &(\mathsf{zero}(G z x_1 x_2 \ldots x_n)) \\
&z \\
&(y(\mathsf{succ}\ z)))\ulcorner 0 \urcorner.
\end{aligned}
$$

Ez a kifejezés azonos a teljes rekurzív függvényekre megadott minimalizáló függvény λ-kifejezésével. Így, ha az $f(x_1, x_2, \ldots, x_n)$ értelmezve van, akkor az *v)* pontban leírtak szerint a $g(z, x_1, x_2, \ldots, x_n)$ is mindig értelmezett, és van olyan y, hogy $g(y, x_1, x_2, \ldots, x_n) = 0$, azaz

$$F\ulcorner x_1 \urcorner \ldots \ulcorner x_n \urcorner = \ulcorner f(x_1, x_2, \ldots, x_n) \urcorner.$$

Ha az $f(x_1, x_2, \ldots, x_n)$ nincs értelmezve, akkor minden z-re

$$g(z, x_1, x_2, \ldots, x_n) \neq 0,$$

és így

$\mathsf{zero}(Gzx_1x_2\dots x_n) = \mathsf{false}.$

Belátható, hogy ekkor az

$$\begin{aligned}\mathsf{Y}(\lambda yz.\mathsf{if}\ &(\mathsf{zero}(Gzx_1x_2\dots x_n))\\ &z\\ &(y(\mathsf{succ}\, z)))\ulcorner 0 \urcorner\end{aligned}$$

kifejezésnek végtelen redukálási sorozata van, és így nincs normál formája, és az is bizonyítható, hogy ez a kifejezés nem megoldható.

Az 1.6.10. tétel tehát kiterjeszthető a parciális numerikus függvények és az 1.7.29. definícióval λ-definiálható függvények kapcsolatára is.

1.7.31. Tétel. (Kleene-tétel [1936.])

A λ-definiálható parciális függvények osztálya pontosan azonos a parciális rekurzív függvények osztályával.

1.8. Konzisztens formális elmélet

Church a λ-kalkulus definiálásával arra törekedett, hogy elméletével minden matematikai meggondolás formalizálható legyen, még azok is, amelyek ellentmondásra vezetnek. Az 1.5.3. pontban, a Russell-paradoxon vizsgálatánál azonban láttuk, hogy a λ-kalkulus általános függvényfogalma ellentmondáshoz vezethet. Ha a λ-kifejezésekhez fogalmakat rendelünk, azaz ha megadjuk a λ-kifejezések szemantikáját, akkor könnyen ellentmondásra juthatunk. Megjegyezzük, hogy ezt a problámát a *típus* bevezetésével, azaz a függvényfogalom általánosságának korlátozásával meg lehet oldani, a *típusos λ-kalkulusban* az ellentmondások elkerülhetők.

Az 1.1.1., 1.2.27. és 1.4.23. definíciók egy (első rendű) *formális elméletet* határoznak meg. A λ-kifejezésekre megadott szintaktikai szabályok alkotják az elmélet nyelvi részét, a formulák az $E = F$ típusú egyenlőségek, és a formulák bizonyítására az 1.2.27. definícióban megadott axiómák szolgálnak. Ha a formulák egyenlőségek, a formális elméletet *egyenlőségi elméletnek* is nevezhetjük. A λ-kalkulus, $\lambda\eta$-kalkulus, $\lambda{+}ext$-kalkulus tehát formális (egyenlőségi) elméletek.

A formális elméletek egyik alapvető problémája az, hogy az elmélet konzisztens-e. A továbbiakban a λ-kalkulus konzisztenciájával foglalkozunk.

1.8.1. Definíció. Konzisztens formális elmélet:

Egy olyan formális elmélet, amelyben a Λ^0 elemei, mint formulák között egyenlőség adható meg, konzisztens, ha van olyan E, $F \in \Lambda^0$, amelyre $E \neq F$.

A definíció tehát azt mondja ki, hogy egy konzisztens formális elméletben nem lehet minden egyenlőséget bizonyítani, azaz ha az $E = F$ nem bizonyítható egyenlőség, akkor az $E = F$-et feltéve a formális elmélet inkonzisztenssé válik.

Most megmutatjuk, hogy a λ-kifejezésekből, konverziós szabályokból álló típusnélküli λ-kalkulus konzisztens.

1.8.1. A $\boldsymbol{\lambda}$-kalkulus konzisztens

Ha az E és F λ-kifejezések normálformája különböző, akkor az 1.3.12. következmény alapján $E = F$ esetén lenne olyan λ-kifejezés, például az E, amelynek két különböző normál formája lenne. Most megmutatjuk, hogy ha az E-nek és az F-nek különböző normál formája van, akkor az $E = F$ bevezetésével a λ-kalkulus inkonzisztenssé válik.

Az állítás a Böhm-tételen alapul. A tétel a λ-kalkulus egyik klasszikus eredménye, jól alkalmazható λ-kifejezések egymástól való megkülönböztetésére.

1.8.2. Tétel. (Böhm-tétel [1968.])

Ha $E, F \in \Lambda^0$ λ-kifejezéseknek van normál formájuk, és $E \neq F$, akkor léteznek olyan $G_1, G_2, \ldots, G_n \in \Lambda^0$ $(n \geq 1)$ λ-kifejezések, amelyekre

$$EG_1G_2 \ldots G_n = \mathsf{false},$$
$$FG_1G_2 \ldots G_n = \mathsf{true}.$$

A tételt nem bizonyítjuk.

Ha a tételben mindkét egyenlet mindkét oldalára a H_1, H_2 λ-kifejezéseket applikáljuk, akkor

$$\mathsf{false}\ H_1H_2 \equiv (\lambda xy.y)H_1H_2 = H_2,$$
$$\mathsf{true}\ H_1H_2 \equiv (\lambda xy.x)H_1H_2 = H_1,$$

és így a tétel egyenletei tetszőleges $H_1, H_2 \in \Lambda^0$ λ-kifejezésekre a következő formában is felírhatók:

$$EG_1G_2\ldots G_nH_1H_2 = H_2,$$
$$FG_1G_2\ldots G_nH_1H_2 = H_1.$$

Most megmutatjuk, hogy két különböző normál formáju λ-kifejezés nem lehet egyenlő:

1.8.3. Következmény.

Ha az E és F λ-kifejezéseknek különböző normál formájuk van és bevezetjük az $E = F$ egyenlőséget, akkor a λ-kalkulus inkonzisztenssé válik.

Bizonyítás. A következmény azt mondja ki, hogy $E = F$ esetén tetszőleges H_1, H_2 egyenlősége bizonyítható. Legyen $G \equiv \lambda xyz.zyx$, és legyenek H_1 és H_2 tetszőleges λ-kifejezések. Ekkor

$$GH_1H_2\ \mathsf{false} = \mathsf{false}\ H_2H_1 = H_1,$$
$$GH_1H_2\ \mathsf{true} = \mathsf{true}\ H_2H_1 = H_2.$$

A Böhm-tétel alapján, ha az E és F λ-kifejezéseknek van normál formájuk, és ezek különbözőek, akkor léteznek olyan $G_1, G_2, \ldots G_n$ λ-kifejezések, melyekre

$$EG_1G_2\ldots G_n = \mathsf{false},$$
$$FG_1G_2\ldots G_n = \mathsf{true},$$

és ezeket a fenti két egyenletbe helyettesítve

$$GH_1H_2(EG_1G_2\ldots G_n) = H_1,$$
$$GH_1H_2(FG_1G_2\ldots G_n) = H_2.$$

Ebből pedig az $E = F$ feltétel miatt $H_1 = H_2$, tehát azt kapjuk, hogy tetszőleges két λ-kifejezés egyenlősége bizonyítható, azaz a λ-kalkulus inkonzisztenssé vált. □

Tehát a különböző normál formájú λ-kifejezések nem lehetnek egyenlők. Olyan két λ-kifejezés, amelyeknek a normál formájuk különböző, nyilvánvalóan van, ebből viszont azonnal következik a λ-kalkulus konzisztens tulajdonsága:

1.8.4. Tétel. (Konzisztencia tétel)

A λ-kalkulus konzisztens.

A λ-kalkulus akkor is inkonzisztenssé válik, ha olyan λ-kifejezéseket kapcsolunk össze az $=$ relációval, amelyeknek nincs normál formájuk:

1.8.5. Következmény.

Ha az E és F λ-kifejezéseknek nincs normál formájuk és bevezetjük az $E = F$ egyenlőséget, akkor a λ-kalkulus inkonszisztenssé válik.

Bizonyítás. Legyen

$$E \equiv \lambda x.x\,\mathsf{K}\,\Omega \equiv \lambda x.x(\lambda xy.x)\Omega,$$
$$F \equiv \lambda x.x\,\mathsf{S}\,\Omega \equiv \lambda x.x(\lambda xyz.xz(yz))\Omega,$$

és tegyük fel, hogy $E = F$. Ekkor

$$E(\lambda xy.x) \equiv (\lambda x.x(\lambda xy.x)\Omega)(\lambda xy.x) = \lambda xy.x,$$
$$F(\lambda xy.x) \equiv (\lambda x.x(\lambda xyz.xz(yz))\Omega)(\lambda xy.x) = \lambda xyz.xz(yz),$$

azaz az $E = F$ miatt

$$\lambda xy.x = \lambda xyz.xz(yz).$$

Ebből tetszőleges G_1, G_2, G_3 λ-kifejezésre

$(\lambda xy.x)G_1G_2G_3 = (\lambda xyz.xz(yz))G_1G_2G_3$, és a redukciókat végrehajtva

$$G_1G_3 = G_1G_3(G_2G_3).$$

Legyen $G_1 \equiv G_3 \equiv \lambda x.x$, ekkor

$(\lambda x.x)(\lambda x.x) = (\lambda x.x)(\lambda x.x)((G_2(\lambda x.x)),$
$\lambda x.x = G_2(\lambda x.x).$

Most legyen $G_2 \equiv (\lambda xy.x)H$, ahol H egy tetszőleges λ-kifejezés. Ekkor

$$G_2(\lambda x.x) \equiv (\lambda xy.x)H(\lambda x.x) = H,$$

és így

$$\lambda x.x = H.$$

Mivel H tetszőleges λ-kifejezés volt, a fenti egyenlőség jobboldalára tetszőleges λ-kifejezés írható, azaz a λ-kalkulus inkonzisztenssé vált. □

1.8.2. A megoldhatóság és a konzisztencia

Az 1.6. pontban láttuk, hogy a parciális függvények a fej normál forma, azaz a megoldhatóság felhasználásával probléma nélkül definiálhatók, a nem megoldható kifejezések a „definiálatlanság" fogalmát valósítják meg. Most megmutatjuk, hogy — ellentétben a normál formával, — ha a nem megoldható kifejezéseket egyenlőeknek tekintjük, a λ-kalkulus konzisztens marad.

Először a helyettesítés műveletét általánosítjuk. Helyettesítéskor ügyelni kell arra, hogy a kifejezés szabad változója nehogy kötötté váljon, ezt a problémát a *kontextus* bevezetésével oldjuk meg.

Ha egy kifejezést úgy építünk fel, hogy benne „üres helyeket", „lyukakat" hagyunk, és ezekre az üres helyekre, feltételek nélkül, tetszőleges kifejezéseket írhatunk be, akkor ezt a kifejezést *kontextusnak* nevezzük. Az üres helyeket az X vagy X_i $(i \geq 1)$ *kontextus változókkal* jelöljük, a kontextus változóinak száma a kontextus *dimenziója.*

Egy n -dimenziós kontextust $\mathcal{C}[X_1, X_2, \ldots, X_n]$ -nel jelölünk. Ha a $\mathcal{C}[X_1, X_2, \ldots, X_n]$ kontextusba rendre az $E_1, E_2, \ldots, E_n$ kifejezéseket helyettesítjük, akkor a $\mathcal{C}[E_1, E_2, \ldots, E_n]$ kifejezést kapjuk.

1.8.6. Példa. *(Kontextus)*
Ha $\mathcal{C}[X] \equiv \lambda x.Xyz$, akkor

$\mathcal{C}[x] \equiv \lambda x.xyz$
$\mathcal{C}[y] \equiv \lambda x.yyz,$
$\mathcal{C}[\lambda y.y] \equiv \lambda x.(\lambda y.y)yz,$
$\mathcal{C}[\lambda y.xy] \equiv \lambda x.(\lambda y.xy)yz.$

Látható, hogy a $\mathcal{C}[x]$ és a $\mathcal{C}[\lambda y.xy]$ kifejezésben az x változó kötötté vált. $\square$

A következő lemma a nem megoldható kifejezések egy fontos tulajdonságát írja le.

1.8.7. Lemma. (Az általánosíthatóság lemmája)

Tegyük fel, hogy E egy nem megoldató kifejezés, és az F -nek van normál formája. Ekkor, ha minden egydimenziós $\mathcal{C}$ kontextusra

$\mathcal{C}[E] = F,$

akkor minden G λ-kifejezésre

$\mathcal{C}[G] = F.$

A lemmát nem bizonyítjuk, a bizonyítás például [Bar84]-ben megtalálható. A lemma azt mondja ki, hogy ha egy kifejezés redukálásakor egy olyan részkifejezést kapunk, ami nem megoldható, akkor, feltéve, hogy a kifejezésnek van normál formája, a redukálások eredményében ez a részkifejezés nem játszik szerepet, helyette ide bármilyen kifejezést beírhatunk. Ez már azt mutatja, hogy a nem megoldható λ-kifejezések nagyon hasonló tulajdonságokkal rendelkeznek.

Először bővítsük a λ-kalkulust a kontextusokra vonatkozó egyenlőségekkel, megmutatjuk, hogy ez az elmélet konzisztens.

1.8.8. Lemma. (A $\mathcal{H}^*$ elmélet)

Jelöljük $\mathcal{H}^*$ -gal azt a formális elméletet, ahol a λ-kifejezésekre a következő egyenlőséget adjuk meg:

$E =_{H^*} F$,

ha minden egydimenziós $\mathcal{C}$ kontextusra

$\mathcal{C}[E]$ *akkor és csak akkor megoldható, ha* $\mathcal{C}[F]$ *is megoldható.*

A $\mathcal{H}^*$ konzisztens.

Bizonyítás. Először azt kell belátnunk, hogy ez az elmélet teljesíti az egyszerű típusnélküli λ-kalkulusra az 1.2.27. definícióban megadott axiómákat.

Az $=_{H^*}$ reláció nyilvánvalóan ekvivalencia reláció, azaz reflexív, szimmetrikus és tranzitív. A μ-szabály a következő: Tegyük fel, hogy $E =_{H^*} F$, és ekkor be kell látni, hogy tetszőleges G kifejezésre $EG =_{H^*} FG$, azaz hogy tetszőleges $\mathcal{D}$ kontextusra $\mathcal{D}[EG]$ akkor és csak akkor megoldható, ha $\mathcal{D}[FG]$ is megoldható.

Nyilvánvaló, hogy ha $\mathcal{D}[EG]$ megoldható, akkor létezik olyan $\mathcal{C}[X]$ kontextus, melyre $\mathcal{C}[E]$ is megoldható, és fordítva, ha $\mathcal{C}[E]$ megoldható, akkor létezik olyan $\mathcal{D}[X]$ kontextus, melyre $\mathcal{D}[EG]$ is megoldható.

$$
\begin{aligned}
&\mathcal{D}[EG]\text{megoldható} \Longleftrightarrow \\
&\mathcal{C}[E]\text{megoldható} \Longleftrightarrow \qquad\qquad\qquad (E =_{H^*} F) \\
&\mathcal{C}[F]\text{megoldható} \Longleftrightarrow \\
&\mathcal{D}[FG]\text{megoldható},
\end{aligned}
$$

így $EF =_{H^*} EG$. A $GE =_{H^*} GF$ és a $\lambda x.E =_{H^*} \lambda x.F$ hasonló módon bizonyítható.

Most mutatjuk meg, hogy az új egyenlőség reláció a β-konverzió bővítése.

Tegyük fel, hogy $E =_\beta F$. Legyen $\mathcal{C}$ egy tetszőleges olyan kontextus, melyre $FV(E) \cup FV(\mathcal{C}) \subseteq \{x_1, x_2, \ldots, x_n\}$. Ekkor

$\mathcal{C}[E]$ megoldható $\Longleftrightarrow$
$\lambda x_1 x_2 \ldots x_n.\mathcal{C}[E]$ megoldható $\Longleftrightarrow$
$\exists G_1 G_2 \ldots G_n$, melyre $(\lambda x_1 x_2 \ldots x_n.\mathcal{C}[E]) G_1 G_2 \ldots G_n = \mathsf{I} \Longleftrightarrow$
$\exists G_1 G_2 \ldots G_n$, melyre $(\lambda x_1 x_2 \ldots x_n.\mathcal{C}[F]) G_1 G_2 \ldots G_n = \mathsf{I} \Longleftrightarrow$
$\lambda x_1 x_2 \ldots x_n.\mathcal{C}[F]$ megoldható $\Longleftrightarrow$
$\mathcal{C}[F]$ megoldható,

azaz $E =_{H^*} F$.

A konzisztencia 1.8.1. definíciójának megfelelően, adunk két olyan kifejezést, amelyeknek az egyenlősége nem bizonyítható. Legyen

$\mathcal{C}[X] \equiv X$,
$E \equiv \mathsf{I}$,
$F \equiv \Omega$.

Tegyük fel, hogy $E =_{H^*} F$, azaz minden $\mathcal{C}$ kontextusra $\mathcal{C}[E]$ akkor és csak akkor megoldható, ha $\mathcal{C}[F]$ is megoldható. $\mathcal{C}[E] \equiv \mathcal{C}[\mathsf{I}] \equiv \mathsf{I}$, tehát E nyilvánvalóan megoldható, és $\mathcal{C}[F] \equiv \mathcal{C}[\Omega] \equiv \Omega$, amiről az 1.7.8. példában láttuk, hogy nem megoldható, azaz csak az $\mathsf{I} \neq_{H^*} \Omega$ egyenlőtlenség állhat fenn. □

Szűkítsük a $\mathcal{H}^*$-ra megadott relációt.

1.8.9. Lemma. (A $\mathcal{H}$ elmélet)

Jelöljük $\mathcal{H}$ -val azt a formális elméletet, ahol a λ-kifejezésekre a következő egyenlőséget adjuk meg:

$E =_H F$,

ha $E, F \in \Lambda^0$ nem megoldható kifejezések.

A $\mathcal{H}$ konzisztens.

Bizonyítás. Bebizonyítjuk, hogy $\mathcal{H} \subseteq \mathcal{H}^*$. Legyen $E, F \in \Lambda^0$ két nem megoldható kifejezés, azaz legyen $E =_H F$, megmutatjuk, hogy ekkor az $E =_{H^*} F$ egyenlőség is fennáll.

Legyen $\mathcal{C}$ egy olyan kontextus, melyekre $FV(E) \cup FV(E) \cup FV(\mathcal{C}) \subseteq$

$\{x_1, x_2, \ldots, x_n\}$. Ekkor

$\mathcal{C}[E]$ megoldható $\Longleftrightarrow$
$\lambda x_1 x_2 \ldots x_n.\mathcal{C}[E]$ megoldható $\Longleftrightarrow$
$\exists G_1 G_2 \ldots G_n$, melyre $(\lambda x_1 x_2 \ldots x_n.\mathcal{C}[E]) G_1 G_2 \ldots G_n = \mathsf{I}$.

Az általánosíthatóság lemmája alapján

$\exists G_1 G_2 \ldots G_n$, melyre $(\lambda x_1 x_2 \ldots x_n.\mathcal{C}[F]) G_1 G_2 \ldots G_n = \mathsf{I} \Longleftrightarrow$
$\lambda x_1 x_2 \ldots x_n.\mathcal{C}[F]$ megoldható $\Longleftrightarrow$
$\mathcal{C}[F]$ megoldható,

azaz $E =_{H^*} F$.

A I és az Ω kifejezések nyilván nem egyenlőek, így $\mathcal{H}$ konzisztens formális elmélet. □

Tehát megállapíthatjuk, hogy a λ-kalkulusban a nem megoldható kifejezések, azaz amelyeknek nincs fej normál formájuk, egyenlőeknek tekinthetők, egy „értéket” képviselnek.

1.8.3. Eldönthetetlen problémák

A kérdés ezek után az, hogyan lehet két λ-kifejezésről eldönteni, hogy vajon egyenlők, azaz egymásba konvertálhatók-e.

Ez a probléma azonban algoritmikusan eldönthetetlen. Egy kérdést *algoritmikusan eldönthetőnek* nevezünk, ha megadható hozzá egy olyan algoritmus, amely a kérdésre véges lépésben „igen” vagy „nem” választ ad. Tehát az egymásba konvertálhatóság eldönthetetlensége azt jelenti, hogy nem létezik olyan eljárás, amellyel eldönthető lenne, hogy egy tetszőleges, adott λ-kifejezés egy másik adott λ-kifejezésbe konvertálható-e.

A λ-kifejezések egy $\mathcal{A}$ halmazát *a konverziós szabályokra nézve zártnak* nevezzük, ha $E \in \mathcal{A}$ és $E = F$ esetén az $F \in \mathcal{A}$ is teljesül.

1.8.10. Tétel. (Scott–Curry eldönthetetlenségi tétele)

Legyen $\mathcal{A}$ és $\mathcal{B}$ λ-kifejezéseket tartalmazó, a konverziókra zárt két nem üres halmaz, ekkor nincs olyan G λ-kifejezés, amelyre

$$G\ulcorner\#E\urcorner = \begin{cases} \ulcorner 1\urcorner, & ha\ E \in \mathcal{A}, \\ \ulcorner 0\urcorner, & ha\ E \in \mathcal{B}. \end{cases}$$

Bizonyítás. A bizonyítás indirekt, tegyük fel, hogy az $\mathcal{A}$ és $\mathcal{B}$ zárt, nem üres halmazokhoz található egy, a tételben megadott tulajdonságú G λ-kifejezés. Megmutatjuk, hogy ebből ellentmondásra jutunk.

Legyen D egy olyan λ-absztrakció, amely tetszőleges x és y λ-kifejezésre a

$$Dxy\ulcorner 1\urcorner = y,$$
$$Dxy\ulcorner 0\urcorner = x$$

eredményt adja. Ilyen D λ-kifejezés lehet például a

$$\lambda xyz.\mathsf{if}(\mathsf{zero}\ z)x\ y,$$

vagy a

$$\lambda xyz.z(\mathsf{true}\ y)x \equiv \lambda xyz.z(\lambda v.y)x.$$

Legyen

$$H \equiv \lambda z.Dxy(G(\mathsf{app}\ z(\mathsf{num}\ z))),$$

ahol az app és a num az 1.5.7. tétel bizonyításában szereplő λ-kifejezések. Legyen továbbá

$$J \equiv H\ulcorner\#H\urcorner.$$

Ekkor

$$J \equiv H\ulcorner\#H\urcorner \equiv$$
$$(\lambda z.Dxy(G(\mathsf{app}\ z(\mathsf{num}\ z))))\ulcorner\#H\urcorner \twoheadrightarrow_\beta$$
$$Dxy(G(\mathsf{app}\ulcorner\#H\urcorner(\underline{\mathsf{num}\ulcorner\#H\urcorner}))) =$$

$Dxy(G(\underline{\mathsf{app}\ulcorner\#H\urcorner\ulcorner\#H\urcorner\urcorner})) =$
$Dxy(G\ulcorner\#(H\ulcorner\#H\urcorner)\urcorner) \equiv$
$Dxy(G\ulcorner\#J\urcorner)$,

tehát

$J = Dxy(G\ulcorner\#J\urcorner)$.

Ha $J \in \mathcal{A}$, akkor a G λ-kifejezés definíciója alapján

$G\ulcorner\#J\urcorner = \ulcorner 1\urcorner$,

azaz

$J = Dxy\ulcorner 1\urcorner = y$,

vagyis a $\mathcal{A}$ zártsága miatt csak $y \in \mathcal{A}$ állhat fenn. Az y azonban tetszőleges λ-kifejezés, így $y \in \mathcal{B}$ esetén ellentmondásra jutunk.

Hasonlóan, ha $J \in \mathcal{B}$, akkor a G λ-kifejezés definíciója alapján

$G\ulcorner\#J\urcorner = \ulcorner 0\urcorner$,

azaz

$J = Dxy\ulcorner 0\urcorner = x$,

vagyis a $\mathcal{B}$ zártsága miatt csak $x \in \mathcal{B}$ lehetséges, és így ebben az esetben is ellentmondásra jutunk. □

Ha az $\mathcal{A}$ és $\mathcal{B}$ zárt halmazokra létezik a tételben szereplő tulajdonságú G λ-kifejezés, akkor az $\mathcal{A}$ és $\mathcal{B}$ halmazokat *rekurzívan elkülöníthető halmazoknak* nevezzük. A tétel tehát azt mondja ki, hogy a konverziókra nézve zárt halmazok nem rekurzívan elkülöníthetők.

Ha az $\mathcal{A}$ halmaz és az $\mathcal{A}$ komplemens halmaza rekurzívan elkülöníthető, akkor az $\mathcal{A}$ halmazt *rekurzív halmaznak* nevezzük.

Ha a Scott–Curry eldönthetetlenségi tételben a $\mathcal{B}$ halmazt az $\mathcal{A}$ komplemens halmazának választjuk, akkor a következő állítást kapjuk:

1.8.11. Következmény.

Ha az $\mathcal{A}$ halmaz a konverziós szabályokra nézve zárt halmaz, akkor az $\mathcal{A}$ nem rekurzív.

A Scott–Curry eldönthetetlenségi tételből azonnal következik az is, hogy az $E = F$, azaz a konvertálhatóság is eldönthetetlen:

1.8.12. Következmény.

Nincs olyan G λ-kifejezés, amelyre

$$G\ulcorner\#E\urcorner\ulcorner\#F\urcorner = \begin{cases} \ulcorner 1\urcorner, & \textit{ha } E = F, \\ \ulcorner 0\urcorner, & \textit{egyébként.} \end{cases}$$

Bizonyítás. Az 1.8.10. tétel jelöléseit használva, legyen $\mathcal{A}$ egy nem üres, a konverziókra zárt halmaz, és legyen $\mathcal{B} = \Lambda \setminus \mathcal{A}$. Legyen például $\mathcal{A} = \{E | E = \lambda x.x\}$, azaz tartalmazza azokat a λ-kifejezéseket, amelyeknek a normál formája $\lambda x.x$. Ez az $\mathcal{A}$ halmaz nyilván zárt, és nyilvánvalóan a $\mathcal{B}$ is zárt lesz. A Scott-Curry eldönthetetlenségi tétel alapján az $\mathcal{A}$ és $\mathcal{B}$ halmazok rekurzívan nem elkülöníthetők, azaz nincs olyan függvény, amely a $\lambda x.x$ λ-kifejezéssel azonos normál formájú λ-kifejezésekre ugyanazt a függvényértéket adná. $\square$

A második fixpont tétel alkalmazásával bizonyítható, hogy egy λ-kifejezés normál formájának meghatározása is eldönthetetlen.

A következő tételt a benne szereplő halt függvény miatt gyakran *a megállási feladat eldönthetetlenségének* is nevezik.

1.8.13. Tétel. (A normál forma eldönthetetlen)

Nincs olyan halt *λ-kifejezés, amelyre*

$$\mathsf{halt}\ulcorner\#E\urcorner = \begin{cases} \mathsf{true}, & \textit{ha } E\textit{-nek van normál formája,} \\ \mathsf{false}, & \textit{egyébként.} \end{cases}$$

Bizonyítás. Tegyük fel, hogy ilyen halt függvény létezik. Legyen

$$D \equiv \lambda x.\mathsf{if}(\mathsf{halt}\ x)\Omega\ulcorner 0\urcorner.$$

Ekkor a 2. fixpont tétel alapján létezik olyan X λ-kifejezés, amelyre

$X = D\ulcorner\#X\urcorner = \mathsf{if}(\mathsf{halt}\ulcorner\#X\urcorner)\Omega\ulcorner 0\urcorner.$

Ha X -nek van normál formája, akkor a halt definícióját felhasználva

$X = \mathsf{if}\ \mathsf{true}\ \Omega\ulcorner 0\urcorner = \Omega,$

de az Ω-nak nincs normál formája. Ha azt tesszük fel, hogy az X -nek nincs normál formája, akkor az előzőhöz hasonlóan

$X = \mathsf{if}\ \mathsf{false}\ \Omega\ulcorner 0\urcorner = \ulcorner 0\urcorner,$

a $\ulcorner 0\urcorner$ λ-kifejezésnek viszont van normál formája. Tehát mindkét esetben ellentmondásra jutottunk. □

Bizonyítás nélkül kimondjuk a következő állítást is.

1.8.14. Tétel. (A fej normál forma eldönthetetlen)

Nincs olyan E λ-kifejezés, amelyre

$$E\ulcorner\#F\urcorner = \begin{cases} \mathsf{true}, & \textit{ha } F \textit{ -nek van fej normál formája,} \\ \mathsf{false}, & \textit{egyébként.} \end{cases}$$

Az utóbbi két tétel tehát azt mondja ki, hogy a normál formák halmaza és a fej normál formák halmaza, azaz a megoldhatóság nem rekurzív.

A. FÜGGELÉK

Definíciók, tételek jegyzéke

1.1.1. Az egyszerű típusnélküli λ-kifejezések 2
1.1.2. λ-absztrakció 4
1.1.3. Az applikáció 4
1.1.7. A szabad változó 6
1.1.8. A kötött változó 6
1.1.12. λ-kifejezés részkifejezése 9
1.1.14. Zárt λ-kifejezés 9
1.1.16. λ-kifejezés lezárása 10
1.1.18. Helyettesítés 10
1.1.21. Egyszerű helyettesítések 11
1.1.22. Az $E[x := F]$ szabad változói 12
1.1.23. A helyettesítési lemma 12
1.2.1. A β-redukció 14
1.2.2. Leibniz-szabály 14
1.2.4. ξ-szabály 15
1.2.8. A β-redukció és a szabad változók 15
1.2.9. β-redukciók és helyettesítések 16
1.2.11. Az α-konverzió 17
1.2.16. Az α-konverzió és a szabad változók 19
1.2.17. Egyszerű α-konverziók 19
1.2.18. Az α-konverzió és a helyettesítés 19
1.2.22. Két λ-kifejezés egyenlősége 25
1.2.23. Leibniz-szabály 26

1.2.25. ξ-szabály 26
1.2.27. A λ-kalkulus 27
1.2.28. Az η-konverzió 28
1.2.30. A $\lambda\eta$-kalkulus 29
1.2.31. Kiterjeszthetőség 29
1.2.32. A λ+ext-kalkulus 30
1.2.33. Curry-tétel 31
1.3.1. Normál forma 31
1.3.2. Normál formák tétele, Curry-tétel 31
1.3.9. Az I. Church–Rosser-tétel 34
1.3.14. A II. Church–Rosser-tétel 38
1.4.21. A δ-redukció 60
1.4.23. A (konstansos) típusnélküli λ-kifejezés 61
1.5.1. A λ-kifejezés fixpontja 62
1.5.3. Az 1. fixpont tétel 64
1.5.4. Dupla fixpont tétel 65
1.5.5. Többszörös fixpont tétel 65
1.5.6. Gödel-szám 67
1.5.7. A 2. fixpont tétel 67
1.5.10. Fixpont-kombinátorok és a redukció 70
1.6.1. λ-definiálható függvény 73
1.6.2. Primitív rekurzív függvények 74
1.6.8. Teljes rekurzív függvények 77
1.6.10. Kleene-tétel 80
1.6.11. λ-definiálható rekurzív függvény 81
1.7.1. Megoldható λ-kifejezés 82
1.7.11. Megoldhatóság és az absztrakció 86
1.7.12. Megoldhatóság és az applikáció 86
1.7.13. A nem megoldhatóság 86
1.7.14. Az applikációs λ-kifejezés 87
1.7.15. Az absztrakciós λ-kifejezés 87
1.7.16. Fej normál forma 88
1.7.22. Wadsworth-tétel 90
1.7.23. Fej normál forma és az absztrakció 91

1.7.24. Fej normál forma és az applikáció 91
1.7.25. Fej normál forma és a helyettesítés 91
1.7.26. Gyenge fej normál forma 92
1.7.29. λ-definiálható függvény 94
1.7.30. Parciális rekurzív függvények 95
1.7.31. Kleene-tétel 99
1.8.1. Konzisztens formális elmélet 100
1.8.2. Böhm-tétel 100
1.8.4. A λ-kalkulus konzisztens 102
1.8.7. Az általánosíthatóság lemmája 104
1.8.8. A $\mathcal{H}^*$ elmélet 104
1.8.9. A $\mathcal{H}$ elmélet 106
1.8.10. Scott–Curry eldönthetetlenségi tétele 108
1.8.13. A normál forma eldönthetetlen 110
1.8.14. A fej normál forma eldönthetetlen 111

Irodalom

[Bar84] Hendrik Pieter Barendregt, *The Lambda Calculus, Its Syntax and Semantics*, North-Holland, 1984.

[Hin86] J. Roger Hindley, Jonathan P. Seldin, *Introduction to Combinators and λ-Calculus*, Cambridge University Press, 1986.

[Pey87] Simon L. Peyton Jones, *The Implementation of Functional Programming Languages*, Prentice Hall, 1987.

[Dil88] Antoni Diller, *Compiling Functional Languages*, John Wiley & Sons Ltd., 1988.

[Bir88] Richard Bird, Philip Wadler, *Introduction to Functional Programming*, Prentice Hall, 1988.

[Hud89] Paul Hudak, *Conception, Evolution, and Application of Functional Programming Languages*, ACM Computing Surveys, Vol.21, No.3, (September 1989)

[Csö92] Csörnyei Zoltán, *Bevezetés a fordítóprogramok elméletébe, 1. rész [Analízis]*, Tankönyvkiadó, Budapest, 1992.

[Csö93] Csörnyei Zoltán, *Bevezetés a fordítóprogramok elméletébe, 2. rész [Szintézis]*, ELTE Kiadó, Budapest, 1993.

[Fis93] Alice E. Fischer, Frances S. Grodzinsky, *The Anatomy of Programming Languages*, Prentice-Hall International, Inc., 1993.

[Pla93] Rinus Plasmeijer, Marko van Eekelen, *Functional Programming and Parallel Graph Rewriting*, Addison-Wesley Publishing Company, 1993.

[Wil95] Reinhard Wilhelm, Dieter Maurer, *Compiler Design*, Addison-Wesley Publishing Company, 1995.

[Gor96] Mike Gordon, *Introduction to Functional Programming*, Lecture Notes, University of Cambridge, 1996.

[Pau96] Lawrence C. Paulson, *Foundation of Functional Programming*, Lecture Notes, University of Cambridge, 1996.

[Har97] John Harrison, *Introduction to Functional Programming*, Lecture Notes, University of Cambridge, 1997.

[Ong98] C.-H. Luke Ong, *Lambda Calculus*, Lecture Notes, Oxford University, 1998.

[Ker00] Andrew D. Ker, *Lambda Calculus*, Lecture Notes, Oxford University, 2000.

Névmutató

Böhm, 66, 100, 101
Barendregt, 82

Church, 1, 32–35, 38, 54, 58, 67, 72, 73, 76, 81, 85, 99
Curry, 5, 31, 63, 73, 108

de Bruijn, 20, 21, 24

Gödel, 67

Kleene, 1, 80, 98
Klop, 66

Leibniz, 14, 15, 26

Rosser, 1, 34, 35, 38, 58, 73
Russell, 63, 71, 73

Scott, 51, 53, 84, 108

Turing, 64, 75, 78

van der Mey, 66

Wadsworth, 82, 90

Tárgymutató

FV, 6
U_i^n, 74, 95
Z, 74, 95
Λ^0, 9
Λ, 3
μ, 78
app, 67
num, 67
pred, 76
succ, 74, 76, 95
add, 58, 59
and, 41
app, 67
$\mathsf{c_n}$, 54
cons, 46
empty, 48
equal, 59
exp, 58, 59
fac, 62
false, 40
first, 42
halt, 110
head, 47
if, 40, 76
mul, 58, 59
muop, 79
n-es, 43, 45
nil, 46, 48
not, 41
num, 67
or, 41
pair, 42
pred, 49, 51, 56, 57, 76
second, 42
$\mathsf{select_i}$, 44, 45
sub, 59
succ, 48, 49, 51, 54, 55, 76
tail, 47
true, 40
zero, 48, 49, 51, 53, 76
@, 7
$:=$, 10
$=$, 25
$=_H$, 106
$=_{H^*}$, 105
$\equiv$, 3
$\twoheadleftarrow\!\!\!\!\twoheadrightarrow_\alpha$, 17
$\leftarrow_\beta$, 14
$\twoheadleftarrow\!\!\!\!\twoheadrightarrow_\beta$, 14
$\rightarrow_\beta$, 14
$\twoheadleftarrow\!\!\!\!\twoheadrightarrow_\eta$, 28
$\sharp$, 66
$[\ ]$, 43
$\langle\ \rangle$, 43
$\lceil\ \rceil$, 67
$\ulcorner\ \urcorner$, 48

{ }, 45
Ω, 32
Θ, 64
I, 5, 32, 49
K, 5, 32
S, 5
Y, 33, 63, 73, 77

absztrakció
  β, 14
absztrakciós
  λ-kifejezés, 87
adekvát számjegyrendszer, 48
aktuális paraméter, 4
  függvény, 4
alapfüggvény, 74
alkalmazott λ-kalkulus, 61
α-konverzió, 17, **1.2.2.**, 17–19
  elkerülése, **1.2.3.**, 20–24
applikáció, 2, 4, **1.1.2.**, 4–6, 61
  függvény, 4
applikációs
  λ-kifejezés, 87
applikatív sorrendű
  redukálási stratégia, 39
aritmetikai
  művelet, **1.4.4.**, 48–60
axióma
  $\lambda{+}ext$-kalkulus, 30
  $\lambda\eta$-kalkulus, 29
  λ-kalkulus, 27–28
azonosság, 3

balasszociatív, 5, 11

β-absztrakció, 14
β-konverzió, 14, **1.2.1.**, 14–17
β-redukció, 14

Church-számjegyek, 54
csúcs
  redex, 93
currying, 5

de Bruijn
  forma
    λ-kifejezés, 20
  szám, 20
δ-függvény, 60
δ-redukció, **1.4.5.**, 60–61
dimenzió
  kontextus, 103

egyenlőség, **1.2.4.**, 24–27
egyszerű-számjegyek, 49
eldönthetetlen probléma, **1.8.3.**, 107–111
  algoritmikusan, 107
elmélet
  egyenlőségi, 100
  formális, 99
equivalencia reláció, 3, 25
érték szerinti paraméterátadás, 39
η-konverzió, **1.2.6.**, 28–31

fej
  normál forma, **1.7.2.**, 87–91
    gyenge, **1.7.3.**, 91–94
  redex, 88
  változó, 88

fixpont, 62
  kombinátor, 63, **1.5.1.**, 63–68
formális
  elmélet, 99
    konzisztens, 99–111
  paraméter
    λ-absztrakció, 4
függvény, 73
  aktuális paramétere, 4
  alap, 74
  δ, 60
  konstans, 40–61
  λ-definiálható, **1.6.**, 73–82, 94–99
  lépés, 74
  magasabbrendű, 5
  parciális rekurzív, 95
  primitív rekurzív, 74
  rekurzív, 68–71
  teljes rekurzív, 77
  Turing-kiszámítható, 75, 78
függvényapplikáció, 4

Gödel-szám, 67
gráf
  λ-kifejezés, 7
gyenge
  fej normál forma, **1.7.3.**, 91–94
  kiterjeszthetőség szabálya, 31

halmaz
  rekurzív, 109
  rekurzívan elkülöníthető, 109
  zárt, 107
hatáskör, 4
helyettesítés, **1.1.4.**, 10–13, 74, 95
hivatkozási távolság, 20

if-then-else, 39, 40, 76, 78
inverzió
  lásd minimalizálás 77, 95

jobbasszociatív, 4

kalkulus
  λ, 2, 27, 100
  λ+ext, 30, 100
  $\lambda\eta$, 29, 100
kifejezés
  λ-kalkulus, 2
  λ_I-kalkulus, 28
  redukálható, 14, 31
kiterjeszthetőség, **1.2.6.**, 28–31
kombinátor, 9
  fixpont, 63–68
  paradoxon, 63, 73
konfluens
  λ-kalkulus, 36
konstans, **1.4.**, 40–61
  függvény, **1.4.**, 40–61
  logikai, 40–42
  szám, 48–60
konstansos
  típusnélküli
    λ-kalkulus, 61
    λ-kifejezés, 61
kontextus, 103
  dimenzió, 103

változó, 103
konvenció
változókra, 17
konvertálható λ-kifejezések, 25
konverzió
α, 17–19
β, 14–17
η, 28–31
konzisztens
formális elmélet, **1.8.**
formális elmélet, 99–111
λ-kalkulus, 36, 100–107
kötött változó, 6
ξ-szabály, 15, 26

λ+ext-kalkulus, 30, 100
λ-absztrakció, 2, **1.1.1.**, 3–4, 61
aktuális paramétere, 4
formális paramétere, 4
törzse, 4
változója, 4
λ-definiálható függvény, 73, **1.6.**, 73–82, 94, **1.7.4.**, 94–99
$\lambda\eta$-kalkulus, 29, 100
λ_I-kifejezés, 28
λ-kalkulus, **1.**, 1–111
alkalmazott, 61
axiómái, **1.2.5.**, 27–28
konfluens, 36
konstansos, 61
konzisztens, 11, 36, **1.8.1.**, 100–103, **1.8.2.**, 103–107
operációs szemantikája, **1.2.**, 13–31
szintaktikája, **1.1.**, 2–13
típusnélküli, 61
típusos, 99
λ-kifejezés, 2
absztrakciós, 87
applikációs, 87
de Bruijn forma, 20
egyszerű típusnélküli, 2
fixpontja, 62
gráfja, 7
konstansos típusnélküli, 61
konvertálható, 25
lezárása, 10
megoldható, 82
megoldható, 82–86
zárt, 9
λ-term, *lásd* λ-kifejezés
láncolt lista, *lásd* lista
lépésfüggvény, 74
lezárás
λ-kifejezés, 10
lista, 45, **1.4.3.**, 45–48
nil, 46
fejelem, 46
maradék, 47
művelet, **1.4.3.**, 45–48
logikai
konstans, **1.4.1.**, 40–42
művelet, **1.4.1.**, 40–42
lusta paraméterkiértékelés, 37

magasabbrendű függvény, 5
megoldható λ-kifejezés, **1.7.1.**, 82–86

megoldhatóság, **1.7.**, 82–99, 103–107
minimalizálás, 77, 95
mnemonik, 3
mohó paraméterkiértékelés, 39
művelet
  aritmetikai, 48–60
  lista, 45–48
  logikai, 40–42
  n-es, 42–45

n-es, **1.4.2.**, 42–45
  művelet, **1.4.2.**, 42–45
névkonfliktus
  változók, 17
névszerinti paraméterátadás, 37
normál
  forma, **1.3.**, 31–39
    fej, **1.7.2.**, 87–91
    gyenge fej, **1.7.3.**, 91–94
  sorrendű
    redukálási stratégia, 37
normalizáló
  redukálási stratégiá, 36
normált számjegyrendszer, 48

operációs szemantika
  λ-kalkulus, 13–31
optimalizált kiértékelés, 41

paradoxon
  kombinátor, 63, 73
  Russell, 71–73
paraméter
  aktuális, 4
  formális, 4
paraméterátadás
  érték szerinti, 39
  névszerinti, 37
paraméterkiértékelés
  lusta, 37
  mohó, 39
parciális rekurzív függvény, **1.6.3.**, 80–82, 95
precedencia, 5
primitív
  rekurzió, 74, 95
  rekurzív függvény, 74, **1.6.1.**, 74–77

redex, 14, 31
  csúcs, 93
  fej, 88
redukálási stratégia, **1.3.1.**, 33–39
  applikatív sorrendű, 39
  normál sorrendű, 37
  normalizáló, 36
redukálható kifejezés, 14, 31
redukció
  β, 14
  δ, 60–61
reflexív reláció, 3, 13, 26
rekurzió, **1.5.**, 61–73
  primitív, 74, 95
rekurzív
  függvény, **1.5.2.**, 68–71
  halmaz, 109

rekurzívan elkülöníthető halmaz, 109
reláció
  equivalencia, 3, 25
  reflexív, 3, 13, 26
  szimmetrikus, 3, 13, 26
  tranzitív, 3, 13, 26
rendezett pár, 42
részkifejezés, 9
rombusz-tulajdonság, 34
Russell-paradoxon, **1.5.3.**, 71–73

Scott-számjegyek, 51
szabad
  és kötött változók, **1.1.3.**, 6–10
  változó, 6
szabály
  gyenge kiterjeszthetőség, 31
  ξ, 15, 26
számjegyek
  Church, 54
  egyszerű, 49
  Scott, 51
számjegyrendszer, 48
  adekvát, 48
  normált, 48
számkonstans, **1.4.4.**, 48–60
szemantika
  operációs
    λ-kalkulus, 13–31
szimmetrikus reláció, 3, 13, 26
szintaktika
  λ-kalkulus, 2–13

teljes rekurzív függvény, 77, **1.6.2.**, 77–80
típus, 99
típusnélküli
  konstansos
    λ-kalkulus, 61
típusos
  λ-kalkulus, 99
törzs
  λ-absztrakció, 4
tranzitív reláció, 3, 13, 26
Turing-kiszámítható függvény, 75, 78

változó, 2, 61
  fej, 88
  kontextus, 103
  kötött, 6
  λ-absztrakció, 4
  szabad, 6
változók névkonfliktusa, 17

zárt
  halmaz, 107
  λ-kifejezés, 9