Capitolul I

Introducere în l-calcul (lambda-calcul)

l-calcul (sau lambda-calcul) este o teorie a funcţiilor, care iniţial a fost dezvoltată de logicianul Alonzo Church ca bază fundamentală pentru matematică. Această teorie a fost elaborată în anii 1930, cu mult înainte de a fi inventate computerele digitale. Puţin mai devreme (în anii 1920), Moses Schönfinkel a dezvoltata o altă teorie a funcţiilor bazată pe ceea ce numim în zilele noastre teoria "combinatorică".

În anii 30, Haskell Currz a redescoperit şi extins teoria lui Schönfinkel şi a demonstrat (arătat) că era echivalentă cu l-calcul. Referitor la acest lucrum Kleene a arătat că l-calcul a fost un sistem universal de calcul; a fost primul sistem de genul acesta ce a fost riguros analizat.

În anii 50, John McCarthz a fost inspirat de l-calcul, inventând limbajul de programare LISP.

La începutul anilor 60, Peter Landin a arătat (demonstrat) cum pot fi descrise (declarate, specificate) sensurile limbajelor de programare imperativă, prin traducerea (transpunerea) lor în l-calcul. De asemenea, el a inventat un puternic limbaj prototip de programare numit ISWIM [24].

Acesta a introdus principalele notaţii ale programării funcţionale şi a influenţat design-ul pentru ambele limbaje funcţional şi imperativ.

Dezvoltând pe această teorie, Christopher Strachey a pus bazele pentru secţiunea importantă a sensurilor semanticii. Întrebări tehnice privind munca (teoriile) lui Strachey, au inspirat pe matematicianul şi logicianul Dana Scott să inventeze teoria domeniilor, care acum este una dintre cele mai importante părţi din ştiinţa teoretică a computerelor.

În timpul anilor 70, Peter Henderson şi Jim Morris au luat în consideraţie lucrarea lui Landin şi au scris un număr important de lucrări, argumentând că, programarea funcţională a constituit un avantaj important pentru programatorii de software. Aproape în aceeaşi perioadă, David Turner a propus ca teoria "combinatorică" a lui Schönfinkel şi Curry ar putea fi folosită la fel ca şi cod al calculatorului pentru a efectua limbaje de programare funcţională.

Asemenea calculatoare puteau exploata proprietăţile matematice ale calculului lambda pentru evaluarea paralelă a programelor.

În timpul anilor 80, mai multe grupuri de cercetare au preluat ideile lui Henderson şi Turner şi au început să lucreze pentru a pune în practică programarea funcţională prin conceperea unor arhitecturi speciale pentru a le susţine, unele dintre ele având multe procesoare.

Noi, în acest fel, vedem că o ramură obscură a logicii matematice stă la baza dezvoltării teoriei limbajului de programare, după cum urmează:

(i) Studiul întrebărilor fundamentale ale calculului
(ii) Design-ul limbajelor de programare
(iii) Semantica limbajelor de programare
(iv) Arhitectura computerelor

1.1. Sintaxa şi semantica lambda-calculului

Lambda-calcul este o notație pentru a defini funcții. Expresiile notației sunt numite *l-expresii* şi fiecare asemenea expresie defineşte o funcție. Se va vedea ulterior cum pot fi folosite funcțiile pentru a reprezenta o largă varietate de date şi structuri de date, inclusiv numere, perechi, liste etc. De exemplu, va fi demonstrat cum o pereche arbitrară de numere (x,y), poate fi reprezentată ca lambda-expresie. Ca o convenție notațională, numele mnemonice sunt evidențiate în **bold** (îngroşat) sau subliniat, în expresii lambda particulare, de exemplu: **1** este lambda-expresie (definită în subcapitolul 2.3), care este folosită pentru rprezentarea numărului unu (1).

Sunt numai 3 feluri de l-expresii:

(i) **variabilele**: x, y, z etc. Funcțiile ce sunt definite de variabile sunt determinate de legătura cu mediul. Legătura este făcută abstract (vezi 3 mai jos). Noi folosim V1, V2, V3 pentru variabile arbitrare.

(ii) **aplicații de funcții** sau **combinații**: dacă E1 şi E2 sunt l-expresii, atunci şi (E1,E2) este tot o l-expresie; indică rezultatul aplicării funcției indicate de E1, funcției indicate de E2. E1 este numită *rator* (de la ope*rator*) şi E2 este numită *rand* (de la ope*rand*). De exemplu, dacă (*m*, *n*) indică o funcție (*De menționat că,* ***sum*** *este o lambda-expresie, în timp ce + este un simbol matematic în metalimbajul ce îl folosim pentru a vorbi despre lambda-calcul*) reprezentând o pereche de numere *m* şi *n* (vezi subcap. 2.2) şi *sum* (suma) indică o funcție l-calcul adițională, atunci aplicația (**sum**(*m*, *n*)), indică (înseamnă) de fapt, suma m+n.

(iii) **noțiuni abstracte (abstrații)**: dacă V este o variabilă şi E este o l-expresie, atunci şi V este lV. E este o abstrație, legată de variabila E (*bound variable E*) şi corpul E (*body E*). Asemenea noțiuni abstractizate indică funcții ce iau ca argument pe *a* şi returnează ca rezultat funcția indicată de E, într-o împrejurare în care legătura variabilei V, indică pe *a*. Mai specific, notația abstractă lV. E indică o funcție ce are ca argument E' şi o transformă într-un lucru indicat de E

[E'/V] (rezultatul substituirii cu E' pentru V în E, vezi subcap. 1.8). De exemplu, lx.**sum**(x,1) indică o funcție de adunare cu o unitate.

Folosind BNF, sintaxa expresiei lambda este doar:

<l-expression> ::= <variable>

| (<l-expression > <l-expression >)

| (l<variable > . <l-expression >)

Dacă V depăşeşte sintaxa class <variable> şi E, E1, E2, … etc. ..... sintaxa class <l-expression >, atunci BNF o simplifică astfel:

E ::=V | (E1 E2) | lV.E

unde: V – variabile, (E1E2) – aplicații şi lV.E – abstractizarea (notații abstracte).

Descrierea a ceea ce înseamnă l-expresie, ce tocmai a fost exemplificată mai sus, este vagă şi intuitivă. Au fost necesari 40 de ani pentru logicieni (Dana Scott, în fapt 32), pentru a o face riguros şi într-un mod folositor. Nu vom intra în detalii despre asta.

Exemplu: (lx. x) implică 'funcția de identitate': ((lx. x) E) = E.

Exemplu: (lx.(lf. (f x))) implică funcția care atunci când se aplică lui E dă (lf. (f x))[E/x], i.e. (lf. (f E)). Aceasta este funcția care când se aplică lui E' dă (f E)[E' / f], cu alte cuvinte (E' E). În acest fel,

((lx. (lf. (f x))) E) = (lf. (f E))

şi

((lx.(lf. (f x))) = (E' E)

Exercițiul 1

Descrieți funcția implicată de (lx. (lz. z)).

Exemplu: Subcapitolul 2.3 descrie cum pot fi numerele reprezentate de lambda-expresii. Presupunem că acest lucru a fost făcut şi că 0, 1, 2 ... sint lambda-expresii ce repretintă respectiv pe 0, 1, 2 ... . Presupunem că de asemenea **add** este o lambda-expresie ce implică o funcție ce îndeplineşte (satisface):

((**add** *m*) *n*) = *m+n*

Atunci (lx. ((**add** 1) *x*)) este o lambda-expresie ce implică sau defineşte funcția ce transformă pe *n* în *1+n*, şi (lx. (ly. ((**add** *x*) *y*))) este o lambda-expresie implicând (definind) funcția ce transformă pe *m* în funcția care aplicată lui *n* dă *m+n*, numită (ly. ((**add** *m*) y)).

Relaționarea dintre funcțiile **sum** (ii) de la începutul acestui subcapitol (pagina 2) şi funcția **add** din exemplu anterior este explicată în subcapitolul 2.5.

1.2. Convenții notaționale

Următoarele convenţii ajută la eliminarea numărului de paranteze ce trebuiesc scrise.

1. Aplicarea funcţiei asociate la stânga, E1, E2 ... En înseamnă ((... (E1 E2)...)En). De exemplu:

E1 E2 înseamnă (E1 E2)

E1 E2 E3 înseamnă ((E1 E2) E3)

E1 E2 E3 E4 înseamnă ((E1 E2) E3) E4)

2. lV. E1 E2 ... En înseamnă (lV. (E1 E2 ... En)). În acest fel scopul lui 'lV' se extinde la dreapta pe cât de mult este posibil.

3. lV1 ...Vn. E înseamnă (lV1. (... .(lVn. E) ...)). De exemplu:

lx y. E înseamnă (lx. (ly. E))

lx y z. E înseamnă (lx. (ly. (l z. E)))

lx y z w. E înseamnă (lx. (ly. (l z. (l w. E))))

Exemplu: lx y. **add** y x înseamnă (lx. (ly. ((**add** y) x))).

1.3. Variabile libere şi variabile legate

O apariţie a unei variabile V într-o lambda-expresie spunem că e *liberă*, dacă nu e implicată în scopul unei 'lV', altfel o numim *legată*.

De exemplu:

(lx. y x) (lx. x y)

liberă liberă

legată legată

1.4. Regulile conversiei

În capitolul 2 este explicat cum pot fi lambda-expresiile folosite pentru a nota (reprezenta) obiecte date ca numere, şiruri etc. De exemplu, o expresie matematică ca (2+3) x 5 poate fi reprezentată ca lambda-expresie şi "rezultatul" acesteia 25, de asemenea poate fi notat (reprezentat) ca lambda-expresie.

Procedura "simplificării" lui (2+3) x 5 = 25 va fi notată printr-o procedura numită *conversie* (sau *reducere*). Regulile lui l-conversie descrise mai jos, sunt foarte generale, totodată atunci când sunt aplicate pentru lambda-expresii ce reprezintă expresii matematice, simluează evaluarea matematică.

Sunt 3 tipuri de l-conversie numite: a-conversie, b-conversie şi h-conversie (originea denumirii acestor notaţii nu este clară). În stabilirea regulilor conversiei, notaţia E[E'/V] este folosită pentru a explica rezultatul substituirii lui E' pentru fiecare apariţie *liberă* a lui V în E. Substituţia este *validă* dacă şi numai dacă nici o variabilă *liberă* din E' devine *legată* în E[E'/V]. Substituirea este descrisă în mai multe detalii în subcapitolul 1.8.

Regulile l-conversiei

• **a-conversie.**

Oricare abstracție a formei lV. E poate fi convertită la lV'. E[V'/V] demonstrează că substituția lui V' pentru V în E este validă.

• **b-conversie.**

Oricare aplicație a formei (lV. E1) E2 poate fi convertită în E1[E2/V],, demonstrează că substituția lui E2 pentru V în E1 este validă.

• **h-conversie.**

Oricare abstracție a formei lV. (E V) în care V nu are apariții libere în E oiate fi redusă la E.

Următoarele notații vor fi folosite:

• E1 $\xrightarrow{a}$ E2 înseamnă E1 a-converge la E2

• E1 $\xrightarrow{b}$ E2 înseamnă E1 b-converge la E2

• E1 $\xrightarrow{h}$ E2 înseamnă E1 h-converge la E2

În subcapitolul 1.4.4. de mai jos această notație este extinsă.

Cea mai imposrtantă conversie este b-conversie; este cea care poate fi folosită pentru a simula evoluția mecanismelor arbitrare. a-conversie este folosită să facă o manipulare tehnică a variabilelor legate şi h-conversia exprimă faptul că două funcții care întotdeauna au aceleaşi soluții, având aceleaşi argumente, sunt egale (vezi subsapitolul 1.7.). În urmaătarele subcapitole, sunt date mai multe explicații şi exemplea celor 3 tipuri de conversie (obs.: 'conversie' şi 'reducere' sunt folosite mai jos ca sinonime).

1.4.1. a-conversie

O lambda-expresie (în mod obligatoriu o abstracție), căreia o a-reducere poate fi aplicată, este numită o a-redex. Termenul de 'redex' este o prescurtare de la 'expresie redusă'. Regula a-conversiei spune că variabilele legate pot fi redenumite, demonstrând că apariția unui conflict de numire nu se întâmplă. (???)

Exemple

lx. x $\xrightarrow{a}$ ly. y

lx. fx $\xrightarrow{a}$ ly. fy

Nu este cazul în care

$\lambda x.\ \lambda y.$ **add** x y $\xrightarrow{\alpha}$ $\lambda y.\ \lambda y.$ **add** y y

pentru că substituția ($\lambda y.$ **add** x y) [y/x] nu este validă, de vreme ce z ce înlocuieşte x devine variabilă legată.

1.4.2. β-conversie

O lambda-expresie (în mod obligatoriu o abstracție) căreia i se poate aplica o β-reducere, este numită o β-redex. Regula β-conversiei este că o evaluarea a unei funcții de apelare într-un limbaj de programare: corpul E1 al funcției λV. E1 este evaluată într-un mediu în care 'fostul parametru' V este legat de 'actualul parametru' E2.

Exemple

$(\lambda x.\ f\ x)\ E \xrightarrow{\beta}$ f E

$(\lambda x.\ (\lambda y.$ **add** x y)) $\underline{3} \xrightarrow{\beta} \lambda y.$ **add** $\underline{3}$ y

$(\lambda y.$ **add** $\underline{3}$ y) $\underline{4} \xrightarrow{\beta}$ **add** $\underline{3}$ $\underline{4}$

Nu este cazul în care

$(\lambda x.\ (\lambda y.$ **add** x y)) (**square** y) $\xrightarrow{\beta} \lambda y.$ **add** (**square** y) y

pentru că substituirea ($\lambda y.$ **add** x y) [(**square** y)/x] nu este validă, de vreme ce z este liber în (**square** y), dar devine legat după substituirea lui x în ($\lambda y.$ **add** x y).

Este necesar ceva practică pentru a parsa lambda-expresii după regulile din subcapitolul 1.3, în aşa fel încât să identifici β-redex-uri. De exemplu, considerăm aplicația

$(\lambda x.\ \lambda y.$ **add** x y) $\underline{3}$ $\underline{4}$.

Punerea în paranteze conform convențiilor se extinde la forma:

$(((\lambda x.\ (\lambda y.\ ((\textbf{add}\ x)\ y)))\ \underline{3})\ \underline{4})$

ce a avut forma

$((\lambda x.\ E)\ \underline{3})\ \underline{4}$

unde

$E = (\lambda y.$ **add** x y)

$(\lambda x.\ E)\ \underline{3}$ este o β-redex şi poate fi redusă la E[$\underline{3}$/x].

1.4.3 η-conversie

O lambda-expresie (în mod necesar o abstractizare) căreia o η-reducere poate fi aplicată, este numită η-redex. Regula η-conversiei exprimă proprietatea că două funcții sunt egale, dacă dau acelaşi rezultate, când sunt luate în considerare aceleaşi argumente. Această proprietate este numită 'extindere' şi este detaliată mai târziu în subcapitolul 1.7. De exemplu, η-conversia garantează (asigură) că $\lambda x.$ (**sin** x) şi **sin** înseamnă de fapt aceeaşi funcție. La general, λV. (E V) indică funcția care când este aplicată unui argument E', returnează (E V)[E'/V]. Dacă V nu apare liber în E, atunci (E V)[E'/V] = (E E') Astfel că, λV.

E V şi E amândouă dau acelaşi rezultat, în speță EE', când este aplicată argumentelor care sint la fel, şi, de aici rezultă aceeaşi funcție.

Exemple:

lx. **add** x → **add**
h
lx. **add** x y → **add** x
h

Nu este cazul că

lx. **add** x x → **add** x
h

pentru că x este liber în **add** x.

1.4.4 Conversii generalizate

Definițiile lui a-conversie, b-conversie şi h-conversie pot fi generalizate după cum urmează:

- E1 → E2 dacă E2 poate fi obținută din E1 prin a-conversie a irucărui subterm.
a
- E1 → E2 dacă E2 poate fi obținută din E1 prin b-conversie a irucărui subterm.
b
- E1 → E2 dacă E2 poate fi obținută din E1 prin h-conversie a irucărui subterm.
h

Exemple:

((lx. ly. **add** x y) 3) 4 → (ly. **add** 3 y) 4
b
(ly. **add** 3 y) 4 → **add** 3 4
b

Prima este o b-conversie în sensul general pentru că (ly. **add** 3 y) 4 este obținut din ((lx. ly. **add** x y) 3) 4 (care ea însăşi nu este o b-redex), reducând subexpresia (lx. ly. **add** x y) 3). Câteodată vom scrie o secvență de conversii ca cele două de mai sus, astfel:

((lx. ly. **add** x y) 3) 4 → (ly. **add** 3 y) 4 → **add** 3 4
b b

Exercițiul 2

Care dintre cele 3 b-reducții de mai jos sunt conversii generalizate (reducerea unei subexpresii) şi care sunt conversii în sensul celui definit de la pagina 4?

(i) (lx. x) 1 → 1
b
(ii) (ly. y) ((lx. x) 1) → (ly. y)1 → 1
b b
(iii) (ly. y) ((lx. x) 1) → (lx. x)1 → 1
b b

În reducerile (ii) şi (iii) din exercițiul de mai sus, una dintre ele începe cu aceeaşi lambda-expresie, dar reducre redex-urile în ordine diferită.

O proprietate importantă a lui b-reducere este aceea că nu contează în ce ordine se fac acestea, una întotdeauna se termină cu aceleați rezultate. Dacă există mai multe *redex*-uri disjuncte într-o expresie, una le poate reduce în paralel. De menționat, totodată, că există câteva secvențe de reducere, posibile să nu se termine niciodată. Acest lucru este explicat mai târziu în legătură cu normalizarea teoremei de la capitolul 2.9. Este una dintre problemele curente şi importante de cercetare pentru a "cincea generație de programare", în a concepe un procesor ce cercetează evoluția paralelă pentru a mări viteza în execuție a programelor funcționale.

1.5. Egalitatea l-expresiilor

Cele 3 reguli de conversie, păstrează înțelesul lambda-expresiilor, dacă E2 poate fi convertită la E2, atunci E1 şi E2 indică (înseamnă) că este aceeaşi funcție. Această proprietate a conversiei ar trebui intuitiv să fie clară. Este posibil să se dea o definiție matematică a funcției indicate de oa lambda-expresie şi pe urmă să dovedim că această funcție nu este schimbată de a, b sau h-conversie. Să facem asta, este surprinzător de dificil [33] şi nu este scopul acestei cărți.

Pur şi simplu, vom *defini* două lambda-expresii ca fiind egale, dacă ele pot fi transformate, una în cealaltă printr-o secvență de *înainte* şi *înapoi* lambda-conversii. Este important, să fim clar înțeleşi, în legătură cu diferența dintre *egalitate* şi *identitate*. Două lambda-expresii sunt *identice*, dacă sunt formate din *exact* aceleaşi secvențe de caractere; sunt *egale*, dacă una poate fi conversită în cealaltă. De exemplu, lx. x este *egală* cu ly. y, dar nu este *identică* cu ea. Se foloseşte următoarea notație:

- E1Λ E2 înseamnă că E1 şi E2 sunt identice.
- E1 = E2 înseamnă că E1 şi E2 sunt egale.

Egalitatea (=) este definită în termeni de identitate (Λ) şi conversie (→, → şi →), după cum urmează.

Egalitatea l-expresiilor

Dacă E şi E' sunt lambda-expresii, atunci, E = E', dacă E Λ E' sau există expresii E1, E2, E3, ... En, ca acestea:

1. E Λ E1
2. E' Λ En
3. Pentru fiecare *i*, de asemenea

a) Ei → Ei+1 sau Ei → Ei+1 sau Ei → Ei+1 sau

b) Ei+1 → Ei sau Ei+1 → Ei sau Ei+1 → Ei.

Exemple

$$(\lambda x.\ x)\ \underline{1} = \underline{1}$$

$$(\lambda x.\ x)\ ((\lambda y.\ y)\ \underline{1}\ ) = \underline{1}$$

$$(\lambda x.\ \lambda y.\ \textbf{add}\ x\ y)\ \underline{3}\ \underline{4}\ = \textbf{add}\ \underline{3}\ \underline{4}$$

Din definiția egalității (=) rezultă:

(i) Pentru oricare E există E = E (egalitatea este *reflexivă*).

(ii) Dacă E = E', atunci E' = E (egalitatea este *simetrică*).

(iii) Dacă E = E' şi E+ = E'', atunci E = E'' (egalitatea este *tranzitivă*).

Dacă o relație este reflexivă, simetrică şi tranzitivă, atunci ea este numită *relație de echivalență*. În acest fel, = (egalitatea) este o relație de echivalență.

O altă proprietate importantă a egalității (=), este că dacă E1 = E2 şi E1' şi E2' sunt două lambda-expresii care diferă în sensul că una conține pe E1 şi cealaltă pe E2, atunci E1' = E2'. Această proprietate este numită *Legea lui Leibnitz*. Este valabilă pentru că aceeaşi secvență a reducerilor pentru a afla din E1 pe E2, poate fi folosită pentru a afla din E1' pe E2'. De exemplu, E1 = E2, atunci aplicând *Legea lui Leibnitz*, lV. E1 = lV. E2.

Este esențial pentru substituțiile din a şi b-reducții, să fie valide. Validarea cerută, respinge, de exemplu, ca lx. (ly. x) să fie a-redusă la ly. (ly. y) (de vreme ce z devine legată după substituirea lui x în ly. x). Dacă această substituire nevalidă a fost permisă, atunci ar fi trebuit urmată de definiția = ca:

lx. ly. x = ly. ly. y

Dar de vreme ce,

(lx. (ly. x)) 1 2 $\xrightarrow[b]{}$ (ly. 1) 2 $\xrightarrow[b]{}$ 1

şi

(ly. (ly. y)) 1 2 $\xrightarrow[b]{}$ (ly. y) 2 $\xrightarrow[b]{}$ 2

vom fi constrânşi să concluzionăm că 1 = 2. În mod general, prin înlocuirile lui 1 şi 2, prin oricare alte expresii, se poate arăta că oricare alte două expresii sunt egale.

Exercițiul 3.

Găsiți un exemplu care arată că dacă substituirea în b-reducere sunt permise să fie non-valide, atunci rezultă că oricare alte două lambda-expresii sunt egale.

Exemplu

Dacă V1, V2, ... Vn sunt toate distincte şi nici una dintre ele nu apare liberă în oricare ar fi E1, E2, ... En, atunci:

(lV1 V2 ... Vn. E) E1 E2 ... En
= ((lV1. (lV2 ... Vn. E)) E1) E2 ... En
→ ((lV2 ... Vn. E) [E1/V1]) E2 ... En
= (lV2 ... Vn. E [E1/V1]) E2 ... En
.....
= E [E1/V1] [E2/V2] ... [En/Vn]

Exercițiul 4

În ultimul exemplu, unde a fost făcută presupunerea precum că V1, V2 ... Vn sunt toate distincte şi că nici una dintre ele nu apare liberă, în oricare dintre E1, E2 ... En?

Exercițiul 5

Găsiți un exemplu care să arate că dacă V1 = V2, atunci, chiar dacă V2 nu este liberă în E1, nu este neapărat cazul pentru

(lV1V2. E) E1 E2 = E [E1/V1] [E2/V2]

Exercițiul 6

Găsiți un exemplu care să arate că dacă V1 ≠ V2, dar V2 apare liberă în E1, atunci nu este neapărat cazul ca

(lV1V2. E) E1 E2 = E [E1/V1] [E2/V2]

1.6 Relația →

În capitolul anterior E1 = E2 a fost definit pentru a însemna că E2 pentru a fi obținut din E1, printr-o secvență de conversii *înainte* sau *înapoi*. Un cau special este când E2 este obținut din E1, folosind numai conversia *înainte*. Aceasta se scrie E1 → E2.

Definiția lui →

Dacă E şi E' sunt lambda-expresii, atunci E → E', dacă E Λ E' sau există expresiile E1, E2, ... En astfel ca:

1. E Λ E1
2. E' Λ En
3. Pentru fiecare *i* avem ori Ei →a Ei+1 sau Ei →b Ei+1 sau Ei →h+ Ei+1

Să observăm că definiția lui → este exact ca definiția =, cu excepția că partea a doua de la punctul 3 lipseşte.

Exercițiul 7

Găsiți E, E' astfel ca E = E', dar nu este cazul ca E → E'.

Exercițiul 8 (foarte greu!)

Arătați că dacă E1 = E2, atunci există E astfek ca E1 → E şi E2 → E. (Această proprietate este numită Teorema lui Church-Rosser. Câteva consecințe ale teoremei sunt discutate în subcapitolul 2.9).

1.7 Extinderea

Presupunem că V nu apare liber în E1 sau E2 şi

E1 V = E2 V

Atunci conform Legii lui Leibnitz

lV. E1 V = lV. E2 V

astfel, prin h-reducere aplicată la ambele părți, vom avea

E1 = E2

Este adesea comod să demonstrăm, că două lambda-expresii sunt egale folosind această proprietate, cu alte cuvinte să demonstrăm că E1 = E2, demonstrând că E1 V = E2 V pentru câțiva V care nu apar liberi în E1 sau E2. Ne vom referi la asemenea demonstrații ca fiind prin *extindere*.

Exercițiul 9

Arătați că

(lf g x. f x (g x)) (lx y. x) (lx y. x) = lx. x

1.8. Substituția

La începutul subcapitolului 1.4 E [E'/V] era definită ca fiind rezultatul substituirii lui E' pentru fiecare apariție liberă a lui V în E. Substituirea s-a spus că este validă, dacă nici o vaiabilă liberă în E', devine legată în E [E'/V]. În definițiile lui a şi b-conversie, s-a menționat că substituirile implicate, trebuie să fie valide. În acest fel, de exemplu, era doar cazul că

(lV. E1) E2 → E1[E2/V]
b

atâta timp cât substituția E1[E2/V] era validă.

Este foarte comod să extindem înțelesul lui E [E'/V], astfel că nu trebuie să ne facem probleme despre validitate.

Aceasta este îndeplinită de definiția de mai jos, care are proprietatea că pentru *toate* expresiile E, E1 şi E2 şi pentru *toate* variabilele V şi V', avem:

(lV. E1) E2 → E1[E2/V] şi lV. E → lV'. E [V'/V]

Pentru a ne asigura că această proprietate este valabilă, E [E'/V] este definită recursiv pe structura lui E, după cum urmează:

E	E[E'/V]
V	E'
V' (unde V ≠ V')	V'
E1 E2	E1[E'/V] E2 [E'/V]
lV. E1	lV. E1
lV'. E1 (unde V ≠ V' şi V' nu este liberă în E')	lV'. E1 [E'/V]
lV'. E1 (unde V ≠ V' şi V' este liberă în E')	lV''. E1 [V''/V'] [E'/V] (unde V'' este o variabilă şi nu este liberă în E' sau E1)

Această definiție particulară a lui E [E'/V] este bazată pe (dar nu identică cu) cea din anexa C din [2].

Pentru a arăta cum funcţionează aceasta considerăm (ly. y x) [y/x]. Cum y este liber y, ultimul caz din tabelul anterior se aplică. Cum z nu apare în y x sau y, avem:

(ly. y x) [y/x] Λ lz. (y x) [z/y] [y/x] Λ lz. (z x) [y/x] Λ lz. z y

În ultima linie din tabelul anterior, alegerea particulară a lui V'' nu este specificată. Oricare variabilă ce nu apare în E' sau E1, va fi suficientă.

O discuție serioasă despre substituție poate fi găsită în cartea lui Hindley şi Seldin [19], unde diferite proprietăţi tehnice sunt enunţate şi demonstrate. Următorul exercițiu este luat din această carte.

Exercițiul 10

Folosiți tabelul de mai sus pentru a rezolva:

(i) (ly. x (lx. x)) [(ly. y x) / x].

(ii) (y (lz. x z)) [(ly. z y) / x].

Este simplu, chiar dacă ia mai mult timp, pentru a demonstra din definiția lui E [E'/V], că tocmai rezultă din aceasta

(lV. E1) E2 → E1 [E2/V] şi lV. E → lV'. E [V'/V]

pentru toate expresiile E, E1 şi E2 şi pentru toate variabilele V şi V'.

În capitolul 3 se va arăta cum teoria combinatorică poate fi folosită pentru a simplifica substituții complexe, în simple operații. În loca de tehnica combinatorică este posibil a se folosi aşa numita *nameless term* (term fără nume) a lui De Bruijn [6]. Ideea lui De Bruijn este că variabilele pot fi gândite ca şi 'pointerii' pentru lambda-expresiile care îi leagă. În loc de 'labeling' (etichetarea) lambda-expresiilor cu nume (cu alte cuvinte variabile legate) şi identificarea prin aceste nume, unul poate indica l cel mai apropiat, prin alocarea unui

număr pentru nivelurile 'upwards' (în sus) ce trebuiesc atinse. De exemplu, lx. ly. x y ar fi reprezentat de ll2 1. Ca un exemplu mult mai complicat, considerăm expresia de mai jos, în care vom indica numărul de nivele, separând o variabilă din l ce este legată de ea.

3
2
lx. ly. x y (ly. x y y)
1 1

În notația lui De Bruijn aceasta este ll2 1 l3 1 1.

O variabilă liberă este o expresie reprezentată de un număr mai mare decât adâncimea lui l de mai sus; diferitelor variabile libere alocându-se diferite numere. De exemplu,

lx. (ly. y x z) x y w

vor fi reprezentate de

l(l 1 2 3) 1 2 4

De vreme ce sunt doar două l anterioare lui 3, acest număr trebuie să indice o variabilă liberă; în mod similar este doar o l anterioară celei de-a doua apariții a lui 2 şi a lui 4, deci acestea de asemenea trebuie să fie variabile libere. Observăm că 2 nu poate fi folosit să reprezinte pe w, de vreme ce acesta a fost folosit pentru a reprezenta variabila liberă y; în acest fel, alegem primul număr disponibilmai mare ca 2 (3 este deja folosit pentru a reprezenta pe z). Trebuie avut grijă pentru a aloca numere suficient de mari variabilelor libere. De exemplu, prima apariție a lui z în lx. z (ly. z) poate fi reprezentată de 2, dar a doua apariție îl cere pe 3, de vreme ce sunt aceleaşi variabile, trebuie folosit 3.

Exemplu

Cu schema lui De Bruijn lx. x (ly. x y y) va fi reprezentată de l1(l2 1 1).

Exercițiul 11

Ce l-expresie este reprezentată de l2(l2)?

Exercițiul 12

Descrieți un algoritm pentru programa reprezentarea lui De Bruijn a expresiei E[E'/V] din reprezentarea lui E şi E'.