Tutorial Pengenalan Lambda Calculus

Raúl Rojas
Freie Universität Berlin
2015
http://www.inf.fu-berlin.de/inst/ag-ki/rojas_home/documents/tutorials/lambda.pdf

Diterjemahkan oleh: Syahrul Findi Ardiansyah
Diperiksa dan diedit oleh: Ade Azurat
untuk pengajaran di lingkungan Fasilkom UI

1. Definisi

Lambda (λ) *calculus* dapat disebut sebagai "bahasa pemrograman universal yang paling sederhana di seluruh dunia". *Lambda calculus* terdiri dari sebuah aturan transformasi (substitusi variabel) dan sebuah skema definisi fungsi. Diperkenalkan pada tahun 1930-an oleh Alonzo Church sebagai sebuah cara untuk memformalkan konsep dari komputasi yang efektif. *Lambda calculus* bersifat universal dalam arti fungsi komputasi apapun dapat diekspresikan dan dievaluasi dengan menggunakan konsep ini. Sehingga ini ekuivalen dengan *Turing Machine*. Namun, *lambda calculus* menekankan kepada penggunaan aturan transformasi dan tidak peduli tentang bagaimana mesin mengimplementasikannya secara aktual. Pendekatannya lebih ke perangkat lunak (*software*) daripada perangkat keras (*hardware*).

Konsep utama dari *lambda calculus* adalah "ekspresi". Sebuah "nama" atau bisa juga disebut dengan "variabel", adalah sebuah pengenal, yg mana dalam tujuannya, dapat disimbolkan dengan huruf *a, b, c*, ... Sebuah ekspresi didefinisikan secara rekursif sebagai berikut:

$$\begin{aligned} < ekspresi > &:= < nama > \mid < fungsi > \mid < aplikasi > \\ < fungsi > &:= \lambda < nama >.< ekspresi > \\ < aplikasi > &:= < ekspresi >< ekspresi > \end{aligned}$$

Sebuah ekspresi dapat diapit oleh tanda kurung supaya lebih jelas, sehingga jika E adalah sebuah ekspresi, maka (E) juga sebuah ekspresi. Satu-satunya simbol yang digunakan adalah *lambda* (λ) dan *dot* (·). Untuk menghindari penggunaan tanda kurung pada ekspresi yang tidak rapih, kami mengadopsi sebuah kebiasaan yaitu mengelompokkan penerapan fungsi dari kiri, sehingga ekspresi

$$E_1E_2E_3...E_n$$

akan dievaluasi dengan menerapkan ekspresi sebagai berikut:

$$(...((E_1E_2)E_3)...E_n)$$

Seperti yang dapat dilihat dari definisi ekspresi yang telah diberikan di atas, yang digunakan sebagai pengenal (*identifier*) adalah ekspresi λ. Contohnya adalah sebagai berikut:

$$\lambda x.x$$

Ekspresi tersebut mendefinisikan fungsi identitas. Huruf setelah simbol λ adalah pengenal dari argumen fungsi tersebut. Ekspresi setelah tanda titik (pada kasus ini adalah sebuah huruf *x*) disebut dengan "badan" dari definisi tersebut.

Fungsi dapat diterapkan kepada ekspresi. Berikut adalah salah satu contoh dari penerapan fungsinya:

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

Fungsi tersebut adalah fungsi identitas yang diterapkan pada *y*. Tanda kurung digunakan supaya lebih jelas dan tidak ambigu. Penerapan fungsi dievaluasi dengan mengganti nilai dari argumen *x* menjadi *y* di dalam badan dari definisi fungsinya, contoh:

$$(\lambda x.x)y = [y/x]x = y$$

Dalam transformasi ini, notasi $[y/x]$ digunakan untuk menandai bahwa semua *x* yang muncul di dalam ekspresi fungsi tersebut akan digantikan oleh *y*.

Nama-nama argumen pada definisi fungsi tidak memiliki arti tersendiri. Nama-nama argumen tersebut hanyalah *placeholder*, sehingga nama-nama argumen tersebut digunakan untuk menunjukkan bagaimana cara menyusun argumen dari fungsi tersebut ketika dievaluasi. Oleh sebab itu

$$(\lambda z.z) \equiv (\lambda y.y) \equiv (\lambda t.t) \equiv (\lambda u.u)$$

dan sebagainya. Simbol "≡" digunakan untuk menunjukkan bahwa ketika $A \equiv B$, *A* hanyalah sinonim dari *B*.

1.1. Variabel Bebas dan Terikat

Di dalam lambda calculus semua nama bersifat lokal pada definisinya. Di dalam fungsi $\lambda x.x$ disebutkan bahwa *x* bersifat terikat karena kemunculannya di dalam badan dari definisinya ditentukan oleh λx. Sebuah nama argumen yang tidak ditentukan oleh sebuah λ disebut variabel bebas. Pada ekspresi

$$(\lambda x.xy)$$

variabel *x* bersifat terikat dan variabel *y* bersifat bebas.

Pada ekspresi

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

huruf x pada badan dari ekspresi pertama bersifat terikat pada fungsi λ yang pertama. Huruf y pada badan dari ekspresi kedua bersifat terikat pada fungsi λ yang kedua dan x bersifat bebas. Sangat penting untuk memperhatikan bahwa huruf x pada ekspresi kedua benar-benar independen terhadap x pada fungsi pertama.

Secara formal dapat disebutkan bahwa sebuah variabel <nama> bersifat bebas dalam sebuah ekspresi jika memenuhi salah satu dari 3 kasus berikut:

- <nama> bersifat bebas pada <nama>.
- <nama> bersifat bebas pada λ<nama_1>.<ekspresi> jika <nama>≠<nama_1> dan <nama> bersifat bebas pada <ekspresi>.
- <nama> bersifat bebas pada E_1E_2 jika <nama> bersifat bebas pada E_1 atau bersifat bebas pada E_2.

Sebuah variabel <nama> bersifat terikat jika memenuhi salah satu dari 2 kasus berikut:

- <nama> bersifat terikat pada λ<nama_1>.<ekspresi> jika <nama>=<nama_1> atau jika <nama> bersifat terikat pada <ekspresi>.
- <nama> bersifat terikat pada E_1E_2 jika <nama> bersifat terikat pada E_1 atau bersifat terikat pada E_2.

Perlu ditekankan bahwa variabel yang sama dapat muncul bebas dan terikat dalam ekspresi yang sama. Pada ekspresi

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

huruf y yang pertama bersifat bebas pada sub-ekspresi yang diberi tanda kurung sebelah kiri, tapi bersifat terikat pada sub-ekspresi sebelah kanan. Oleh karena itu, dia bersifat bebas dan terikat pada ekspresi yang sama.

1.2. Substitusi

Bagian yang lebih membingungkan dari standar lambda calculus adalah ketika pertama kali ditemukan, faktanya kami tidak memberikan nama kepada fungsinya. Setiap kami ingin mengimplementasikan sebuah fungsi, kami menulis keseluruhan definisi fungsinya lalu mengevaluasinya. Namun untuk menyederhanakan notasinya, kami menggunakan huruf kapital, angka, dan simbol lain sebagai sinonim untuk beberapa definisi fungsi. Contohnya fungsi identitas, dapat dilambangkan dengan I yang merupakan sinonim dari $(\lambda x.x)$.

Fungsi identitas yang diterapkan pada dirinya sendiri adalah

$$\mathsf{I}\mathsf{I}\equiv(\lambda x.x)(\lambda x.x)$$

pada ekspresi tersebut huruf x pertama pada badan dari ekspresi yang diberi tanda kurung pertama bersifat independen pada huruf x di dalam badan ekspresi kedua. Ekspresi di atas dapat dituliskan sebagai berikut

$$\mathsf{I}\mathsf{I} \equiv (\lambda x.x)(\lambda z.z)$$

Fungsi identitas yang diterapkan pada dirinya sendiri

$$\mathsf{I}\mathsf{I} \equiv (\lambda x.x)(\lambda z.z)$$

jika dievaluasi adalah sebagai berikut

$$[\lambda z.z/x]x = \lambda z.z \equiv \mathsf{I}$$

yang mana juga merupakan fungsi identitas.

Perlu diperhatikan ketika melakukan substitusi untuk menghindari percampuran variabel-variabel yang bersifat bebas dengan yang bersifat terikat. Pada ekspresi berikut

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

fungsi pada sebelah kiri berisi variabel y yang bersifat terikat, padahal variabel y yang di sebelah kanan bersifat bebas. Substitusi yang salah akan mencampur 2 variabel tersebut dan menghasilkan

$$(\lambda y.yy)$$

Dengan mengganti nama variabel y yang terikat menjadi t, dapat diperoleh

$$(\lambda x.(\lambda t.xt))y = (\lambda t.yt)$$

yang mana jika dievaluasi akan berbeda hasilnya, tapi inilah cara yg benar.

Dengan demikian, jika fungsi $\lambda x.$<ekspresi> diaplikasikan pada E, akan dilakukan substitusi semua variabel x yang bebas pada <ekspresi> dengan E. Jika proses substitusi menghasilkan variabel bebas dari E pada sebuah ekspresi di mana variabel yang muncul bersifat terikat, nama dari variabel terikat tersebut perlu diganti sebelum melakukan substitusi. Contohnya pada ekspresi

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

kami mengasosiasikan argumen x dengan variabel y. Pada badan definisi fungsi berikut

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

hanya huruf x pertama yang bersifat bebas dan bisa disubstitusi. Sebelum melakukan substitusi, nama variabel y perlu diganti untuk menghindari pencampuran variabel yang terikat dengan variabel yang bebas:

$$[y/x](\lambda t.(x(\lambda x.xt))) = (\lambda t.(y(\lambda x.xt)))$$

Dalam urutan reduksi secara normal kami mencoba untuk selalu mengurangi ekspresi paling kiri dari rangkaian aplikasi. Kami melanjutkan sampai tidak ada reduksi lebih lanjut yang memungkinkan.

2. Perhitungan

Kami berekspektasi bahwa sebuah bahasa pemrograman seharusnya bisa melakukan perhitungan aritmatika. Bilangan dapat direpresentasikan dalam *lambda calculus* mulai dari nol (zero) dan menulis "suc(zero)" untuk mewakili 1, "suc(suc(zero))" untuk mewakili 2, dan seterusnya. Dalam *lambda calculus* hanya dapat mendefinisikan fungsi baru. Bilangan akan didefinisikan sebagai fungsi dengan menggunakan pendekatan sebagai berikut: nol dapat didefinisikan sebagai

$$\lambda s.(\lambda z.z)$$

Ini adalah fungsi dari dua argumen *s* dan *z*. Kami akan menyingkat ekspresi tersebut dengan lebih dari satu argumen menjadi

$$\lambda sz.z$$

Dapat dipahami di sini bahwa *s* adalah argumen pertama yang disubstitusi selama melakukan evaluasi dan *z* merupakan argumen kedua. Dengan menggunakan notasi ini, bilangan asli pertama dapat didefinisikan menjadi

$$1 \equiv \lambda sz.\, s(z)$$
$$2 \equiv \lambda sz.\, s(s(z))$$
$$3 \equiv \lambda sz.\, s(s(s(z)))$$

dan seterusnya.

Fungsi pertama yang sangat menarik adalah fungsi *successor*, yang dapat didefinisikan sebagai berikut

$$\mathsf{S} \equiv \lambda wyx.\, y(wyx)$$

Jika fungsi *successor* diterapkan pada fungsi zero akan menghasilkan

$$\mathsf{S0} \equiv (\lambda wyx.\, y(wyx))(\lambda sz.\, z)$$

Pada badan dari ekspresi pertama dilakukan substitusi semua kemunculan variabel *w* dengan ($\lambda sz.z$) yang menghasilkan

$$\lambda yx.\, y((\lambda sz.\, z)yx) \;=\; \lambda yx.\, y((\lambda z.\, z)x) \;=\; \lambda yx.\, y(x) \equiv 1$$

Artinya, diperoleh representasi dari bilangan 1 (ingat bahwa nama variabel dapat tidak memiliki arti sendiri dan dapat diubah-ubah).

Fungsi successor yang diterapkan pada bilangan 1 akan menghasilkan

$$\mathsf{S1} \equiv (\lambda wyx.\, y(wyx))(\lambda sz.\, s(z)) \;=\; \lambda yx.\, y((\lambda sz.\, s(z))yx) \;=\; \lambda yx.\, y(y(x)) \equiv 2$$

Perhatikan bahwa satu-satunya tujuan menerapkan bilangan (λsz.s(z)) pada argumen y dan x adalah mengganti nama variabel-variabel yang digunakan pada definisi dari bilangan tersebut.

2.1. Penjumlahan

Penjumlahan dapat diperoleh secara langsung dengan memperhatikan bahwa badan fungsi sz dari definisi bilangan 1, misalnya, dapat diinterpretasikan sebagai penerapan fungsi s pada z. Jika ingin menjumlahkan misalnya bilangan 2 dan 3, hanya tinggal mengaplikasikan fungsi *successor* 2 kali pada bilangan 3

Mari mencoba fungsi sebagai berikut untuk menghitung 2+3:

$$\mathsf{2S3} \equiv (\lambda sz.\, s(sz))(\lambda wyx.\, y(wyx))(\lambda uv.\, u(u(uv)))$$

Ekspresi pertama di sebelah kiri adalah bilangan 2, ekspresi kedua adalah fungsi *successor*, dan ekspresi ketiga adalah bilangan 3 (nama variabel sudah diganti supaya lebih jelas). Ekspresi tersebut dapat direduksi menjadi

$$(\lambda wyx.\, y((wy)x))((\lambda wyx.\, y((wy)x))(\lambda uv.\, u(u(uv)))) \equiv \mathsf{SS3}$$

SS3 dapat direduksi menjadi S4 = 5.

2.2. Perkalian

Perkalian 2 bilangan *x* dan *y* dapat dihitung dengan menggunakan fungsi berikut:

$$(\lambda xyz.x(yz))$$

Sehingga perkalian 2 dengan 2 adalah:

$$(\lambda xyz.x(yz))22$$

yang mana dapat direduksi menjadi

$$(\lambda z.2(2z))$$

Dapat dilakukan verifikasi dengan mereduksi ekspresi tersebut lebih lanjut, sehingga nantinya akan mendapatkan hasil 4.

3. Kondisional

Kami akan memperkenalkan 2 fungsi yang nilainya dapat disebut sebagai “true”

$$\mathsf{T} \equiv \lambda xy.x$$

dan “false”

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

Fungsi pertama membutuhkan 2 argumen dan mengembalikan nilai argumen pertama, sedangkan fungsi kedua akan mengembalikan nilai argumen kedua.

3.1. Operasi Logika

Sekarang dapat didefinisikan operasi-operasi logika dengan menggunakan representasi dari nilai kebenaran.

Fungsi AND dari 2 argumen dapat didefinisikan sebagai:

$$\wedge \equiv \lambda xy.xy(\lambda uv.v) \equiv \lambda xy.xy\mathsf{F}$$

Fungsi OR dari 2 argumen dapat didefinisikan sebagai:

$$\vee \equiv \lambda xy.x(\lambda uv.u)y \equiv \lambda xy.x\mathsf{T}y$$

Negasi dari sebuah argumen dapat didefinisikan sebagai:

$$\neg \equiv \lambda x.x(\lambda uv.v)(\lambda ab.a) \equiv \lambda x.x\mathsf{FT}$$

Jika fungsi negasi diterapkan pada nilai "true" adalah

$$\neg\mathsf{T} \equiv \lambda x.x(\lambda uv.v)(\lambda ab.a)(\lambda cd.c)$$

yang akan direduksi menjadi

$$\mathsf{TFT} \equiv (\lambda cd.c)(\lambda uv.v)(\lambda ab.a) = (\lambda uv.v) \equiv \mathsf{F}$$

artinya nilai kebenaran dari operasi tersebut adalah "false".

3.2. Pengujian Kondisional

Akan sangat mudah jika di dalam sebuah bahasa pemrograman bisa memiliki fungsi yang akan bernilai *true* jika sebuah bilangan adalah nol, dan sebaliknya. Fungsi Z berikut ini memenuhi kebutuhan tersebut

$$\mathsf{Z} \equiv \lambda x.x\mathsf{F}\neg\mathsf{F}$$

Untuk bisa memahami bagaimana fungsi ini bekerja, catat bahwa

$$0f\,a \equiv (\lambda sz.z)f\,a = a$$

yang artinya, fungsi f jika diaplikasikan 0 kali ke argumen a akan menghasilkan a. Sedangkan jika F diaplikasikan pada argumen apapun akan menghasilkan fungsi identitas

$$\mathsf{F}a \equiv (\lambda xy.y)a = \lambda y.y \equiv \mathsf{I}$$

Sekarang dapat diuji apakah fungsi Z bekerja dengan benar. Jika fungsi Z diterapkan pada 0 akan menghasilkan

$$\mathsf{Z}0 \equiv (\lambda x.x\mathsf{F}\neg\mathsf{F})0 = 0\mathsf{F}\neg\mathsf{F} = \neg\mathsf{F} = \mathsf{T}$$

karena F diaplikasikan 0 kali pada ¬ akan menghasilkan ¬ pula. Jika fungsi Z diterapkan pada bilangan lainnya akan menghasilkan

$$\mathsf{Z}\mathsf{N} \equiv (\lambda x.x\mathsf{F}\neg\mathsf{F})\mathsf{N} = \mathsf{N}\mathsf{F}\neg\mathsf{F}$$

Fungsi F akan diaplikasikan N kali kepada ¬. Tapi jika F diaplikasikan pada apapun akan bersifat identitas, sehingga ekspresi di atas akan mereduksi semua bilangan N yang lebih besar dari 0 menjadi

$$\mathsf{I}\mathsf{F} = \mathsf{F}$$

3.3. Fungsi Pendahulu (*Predecessor*)

Sekarang fungsi pendahulu dapat didefinisikan dengan mengkombinasikan beberapa fungsi yang telah diperkenalkan sebelumnya. Ketika mencari pendahulu dari *n*, strategi umum yang digunakan adalah membuat pasangan (*n*, *n*-1) dan memilih elemen kedua dari pasangan tersebut sebagai hasilnya.

Pasangan (*a*, *b*) dapat direpresentasikan pada *lambda calculus* dengan menggunakan fungsi

$$(\lambda z.zab)$$

Elemen pertama dari ekspresi pasangan tersebut dapat diambil dengan menerapkan fungsi tersebut pada T

$$(\lambda z.zab)\mathsf{T} = \mathsf{T}ab = a$$

serta menerapkan pada fungsi F untuk mengambil elemen kedua

$$(\lambda z.zab)\mathsf{F} = \mathsf{F}ab = b$$

Fungsi berikut dihasilkan dari pasangan (*n*, *n*-1) (yaitu argumen *p* pada fungsi tersebut) kepada pasangan (*n*+1, *n*-1):

$$\Phi \equiv (\lambda pz.z(\mathsf{S}(p\mathsf{T}))(p\mathsf{T}))$$

Sub-ekspresi $p\mathsf{T}$ mengambil elemen pertama dari pasangan *p*. Pasangan baru dibentuk dengan menggunakan elemen ini, yang ditambahkan untuk elemen pertama dari pasangan yang baru dan disalin untuk elemen kedua dari pasangan yang baru.

Pendahulu dari bilangan *n* diperoleh dengan menerapkan *n* kali fungsi Φ pada pasangan (λ.*z*00) lalu memilih bilangan kedua dari pasangan yang baru

$$\mathsf{P} \equiv (\lambda n.n\Phi(\lambda z.z00)\mathsf{F}$$

Perhatikan bahwa dengan menggunakan pendekatan ini, pendahulu dari nol adalah nol. Sifat ini sangat berguna untuk definisi dari fungsi-fungsi yang lainnya.

3.4. Persamaan dan Pertidaksamaan

Dengan menggunakan fungsi pendahulu sebagai komponen utamanya, kini fungsi yang menguji jika sebuah bilangan *x* lebih besar sama dengan sebuah bilangan *y* dapat didefinisikan sebagai berikut:

$$\mathsf{G} \equiv (\lambda xy.\mathsf{Z}(x\mathsf{P}y))$$

Jika fungsi pendahulu diterapkan *x* kali pada *y* akan menghasilkan nol, maka $x \geq y$ bernilai benar.

Jika $x \geq y$ dan $y \geq x$, maka $x = y$. Hal ini akan mengarah kepada definisi berikut dari fungsi E yang menguji dua bilangan adalah sama:

$$\mathsf{E} \equiv (\lambda xy. \wedge (\mathsf{Z}(x\mathsf{P}y))(\mathsf{Z}(y\mathsf{P}x)))$$

Dengan cara yang sama dapat didefinisikan juga fungsi untuk menguji apakah $x > y$, $x < y$, atau $x \neq y$.

4. Rekursi

Fungsi rekursif dapat didefinisikan dalam *lambda calculus* dengan menggunakan fungsi yang memanggil sebuah fungsi *y* lalu meregenerasi dirinya sendiri. Hal ini dapat dipahami dengan lebih baik dengan mempertimbangkan fungsi Y berikut ini:

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

Jika fungsi tersebut diterapkan pada R akan menghasilkan:

$$\mathsf{YR} = (\lambda x.\mathsf{R}(xx))(\lambda x.\mathsf{R}(xx))$$

yang dapat direduksi menjadi:

$$\mathsf{R}((\lambda x.\mathsf{R}(xx))(\lambda x.\mathsf{R}(xx)))$$

tapi hal ini berarti bahwa YR = R(YR), yaitu fungsi R dievaluasi dengan menggunakan pemanggilan rekursif YR sebagai argumen pertama.

Diasumsikan misalnya ingin mendefinisikan fungsi yang menjumlahkan n bilangan natural pertama. Definisi rekursif dapat digunakan, karena $\sum_{i=0}^{n} i = n + \sum_{i=0}^{n-1} i$. Mari menggunakan definisi fungsi R berikut:

$$\mathsf{R} \equiv (\lambda rn.\mathsf{Z}n0(n\mathsf{S}(r(\mathsf{P}n))))$$

Definisi tersebut memberitahu bahwa bilangan *n* telah diuji: jika bilangan tersebut nol maka hasil dari penjumlahannya adalah nol. Jika *n* bukan nol, maka fungsi *successor* diterapkan *n* kali pada pemanggilan rekursif (argumen *r*) dari fungsi yang diterapkan pada pendahulu dari *n*.

Bagaimana cara mengetahui bahwa *r* pada ekspresi di atas adalah pemanggilan rekursif pada R, padahal fungsi pada *lambda calculus* tidak punya nama? Tidak ada yang tahu dan inilah alasan tepat mengapa operator rekursi Y perlu digunakan. Asumsikan misalnya ingin menjumlahkan bilangan dari 0 sampai 3. Operasi yang dibutuhkan dijalankan dengan pemanggilan:

$$YR3 = R(YR)3 = Z30(3S(YR(P3)))$$

Karena 3 tidak sama dengan nol, evaluasinya akan direduksi menjadi

$$3S(YR2)$$

yaitu jumlah bilangan dari 0 sampai 3 sama dengan 3 ditambah jumlah dari bilangan dari 0 sampai 2. Evaluasi rekursif berturut-turut pada YR akan mengarah pada jawaban akhir yang benar.

Perhatikan bahwa pada fungsi yang telah didefinisikan di atas rekursinya akan putus ketika argumennya menjadi 0. Hasil akhirnya akan menjadi

$$3S2S1S0$$

yaitu bilangan 6.

5. Latihan untuk Pembaca

1. Tentukan fungsi "kurang dari" dan "lebih besar dari" dari dua argumen numerik.
2. Tentukan bilangan bulat positif dan negatif menggunakan pasangan bilangan asli.
3. Tentukan penjumlahan dan pengurangan bilangan bulat.
4. Definisikan pembagian bilangan bulat positif secara rekursif.
5. Definisikan fungsi n! = n · (n - 1) · · · 1 secara rekursif.
6. Definisikan bilangan rasional sebagai pasangan bilangan bulat.
7. Tentukan fungsi untuk penjumlahan, pengurangan, perkalian dan pembagian rasio.
8. Tentukan struktur data untuk merepresentasikan list berisi angka.
9. Tentukan fungsi yang mengekstrak elemen pertama dari sebuah list.
10. Tentukan fungsi rekursif yang menghitung jumlah elemen dalam sebuah list.
11. Bisakah anda melakukan simulasi Turing Machine dengan menggunakan lambda calculus?

Referensi

[1] P.M. Kogge, The Architecture of Symbolic Computers, McGraw-Hill, New york 1991, chapter 4.
[2] G. Michaelson, An Introduction to Functional Programming through λ- calculus, Addison-Wesley, Wokingham, 1988.
[3] G. Revesz, Lambda-Calculus Combinators and Functional Programming, Cambridge University Press, Cambridge, 1988, chapters 1-3.