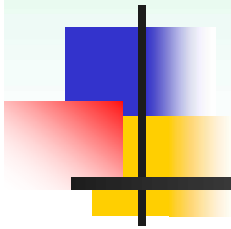
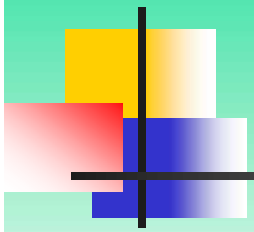


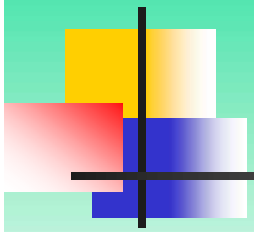
コンピュータよもやま話

コンピュータの基礎



目次

- ハードウェアの話
- カーネルの話
- 形式言語の話
- 関数型言語とラムダ計算の話



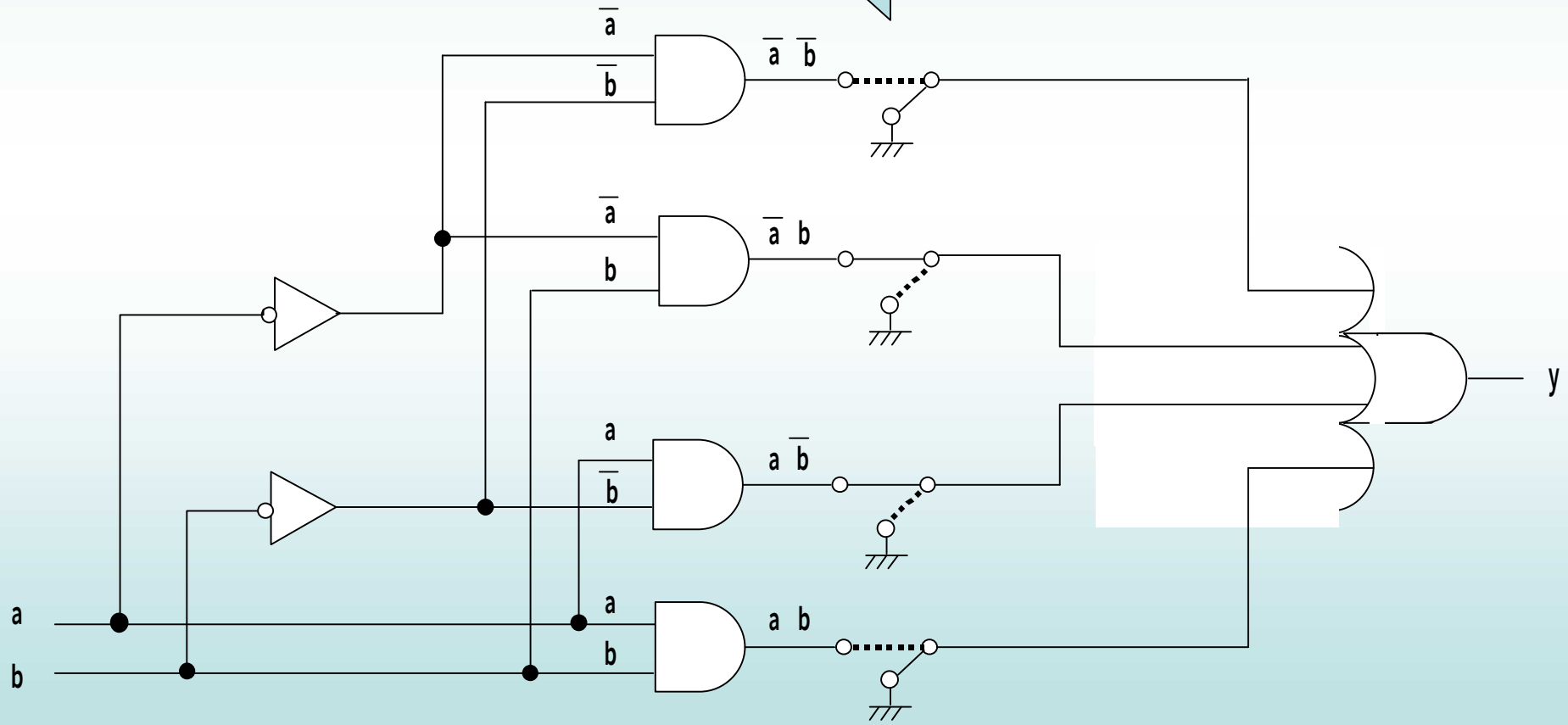
ハードウェアの話

- コンピュータの必須機能を絞り込む
  - フローチャートに従って実行する能力
    - 始点、終点、演算結果の記憶、比較結果に応じた分岐で構成される有向グラフ
- 実現には真理値表から組合せ回路への機械的な変換を利用
  1. NOT、AND、ORから論理演算、四則演算を構成（ALU）
  2. 組合せ回路から記憶回路または順序回路を構成（レジスタ）
  3. プログラムカウンタと条件付きジャンプ（流れの制御）

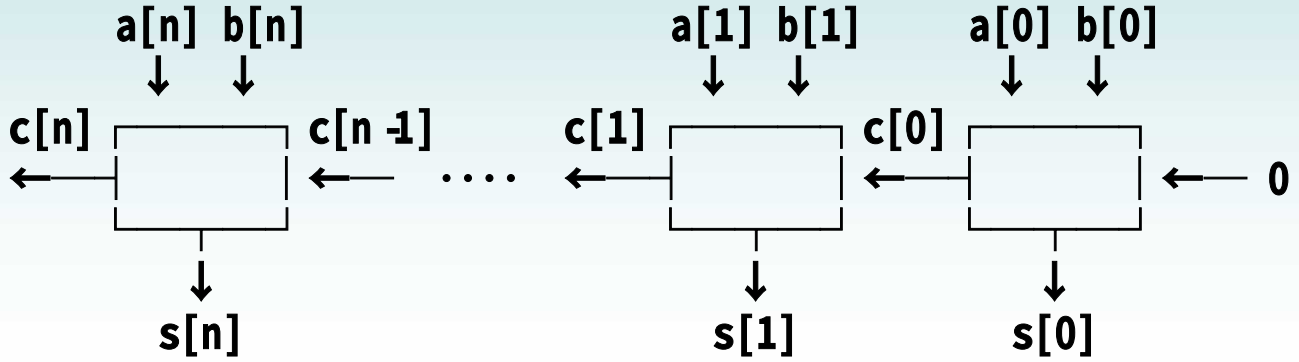
2入力XORの例

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

真理値表から組合せ回路への
機械的な変換



加算 (算術演算)



c[n-1]	a[n]	b[n]	c[n]	s[n]
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

c[n-1] は下位ビットからの
桁上がり(carry)を意味する.

ラムダ計算で論理値や整数値とその演算を表現し組み合わせていく発想と同じ

条件付きジャンプ(流れの制御)

条件付きジャンプが可能になれば条件分岐と繰り返しが可能になる．(チューリング完全につながる)
下図にその原理を示す．正確を期すならばタイミングの説明が必要になるがタイミングチャートは省略する．
ここでは特別なハードウェアが必要ないということを説明したい．

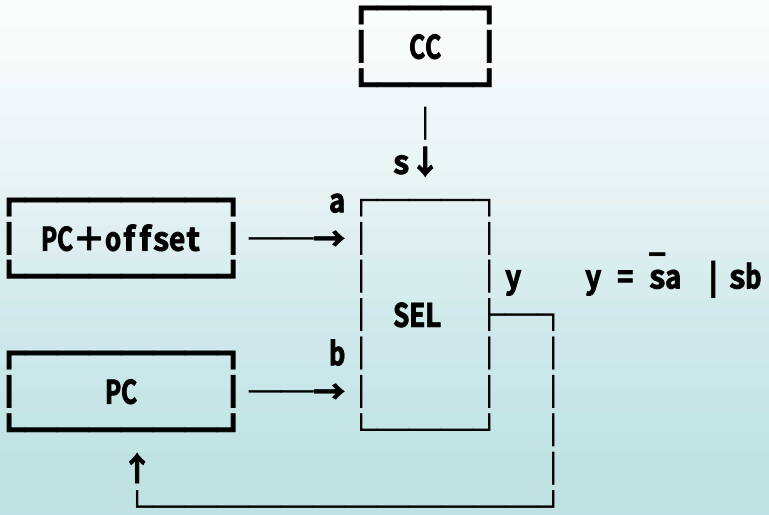
図の説明

PC (Program Counter)
次に実行される命令のアドレスを記憶するレジスタ．自動的に次のアドレスに更新される．
ジャンプ命令実行中はジャンプ命令の次のアドレスを指している．

PC+offset
条件成立時にジャンプする命令のアドレス．

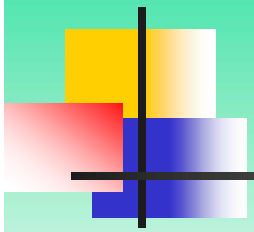
CC (Condition Code Register)
比較命令の演算結果を記憶するレジスタ．
条件付きジャンプ命令の直前には比較命令がありCCの内容はジャンプ先の命令か次の命令かの選択に使われる．

SEL (Selector)
選択信号により入力信号のいずれかを選んで出力するハードウェア．
図では１個しか描かれていないが32ビットならば32個必要になる．



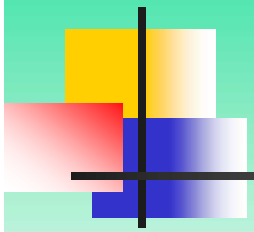
Selectorの真理値表

s	a	b	y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



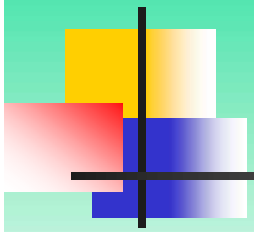
非ノイマン型コンピュータ

- データフロー型コンピュータ
- 要求駆動型コンピュータ



カーネルの話

- プロセスとスレッド
- ファイルシステム
- メモリ管理（仮想記憶）



形式言語の話

- コンパイラとカーネルは特別なプログラム
- チョムスキー階層
  - 句構造文法
  - 文脈依存文法
  - 文脈自由文法
  - 正規文法

  コンパイラは文法による語の生成の逆操作を行うプログラムである。

句構造文法 (phrase structure grammar)

生成規則に制約を設けない文法である.
N:非終端記号　T:終端記号　P:生成規則

文法の例　--a が 2 の巾乗個連結された言語を定義

```
N = { S, A, B, C, D, E }
T = { a }
P = {
      S   -> ACaB,
      Ca  -> aaC,
      CB  -> DB,
      CB  -> E,          この生成規則は左辺よりも右辺の長さが短い
      aD  -> Da,
      AD  -> AC,
      aE  -> Ea,
      AE  -> ε           この生成規則は左辺よりも右辺の長さが短い
    }
開始記号 : S
```

生成過程の例

前述の句構造文法による生成過程の例

S	
→ ACaB	[1] S → ACaB
→ AaaCB	[2] Ca → aaC
→ AaaDB	[3] CB → DB
→ AaDaB	[5] aD → Da
→ ADaaB	[5] aD → Da
→ ACaaB	[6] AD → AC
→ AaaCaB	[2] Ca → aaC
→ AaaaaCB	[2] Ca → aaC
→ AaaaaDB	[3] CB → DB
→ AaaaDaB	[5] aD → Da
→ AaaDaaB	[5] aD → Da
→ AaDaaaB	[5] aD → Da
→ ADaaaaB	[5] aD → Da
→ ACaaaaB	[6] AD → AC
→ AaaCaaaB	[2] Ca → aaC
→ AaaaaCaaB	[2] Ca → aaC
→ AaaaaaaCaB	[2] Ca → aaC
→ AaaaaaaaaCB	[2] Ca → aaC
→ AaaaaaaaaE	[4] CB → E
→ AaaaaaaaEa	[7] aE → Ea
→ AaaaaaaEaa	[7] aE → Ea
• • •	
→ AaEaaaaaaa	[7] aE → Ea
→ AEaaaaaaaa	[7] aE → Ea
→ aaaaaaaa	[8] AE → ε

文脈依存文法 (context sensitive grammar)

文脈依存文法とは句構造文法の生成規則に
**　　右辺は左辺以上の長さでなければならない**
という制約を設けた文法である．

文法の例　-- an・b**n・c**n (a**n は a の n 個連結を表す)**

```
N = { S, B }
T = { a, b, c }
P = {
      S   -> aBSc,
      S   -> abc,
      Ba  -> aB,
      Bb  -> bb
    }
開始記号 : S
```

文脈自由文法 (context free grammar)

生成規則に

$$A \rightarrow \beta$$

という制約を設けた文法である．

ただしＡは非終端記号であり，βは空を除いた非終端記号または終端記号の列である．

左辺は１個の非終端記号でなければならないことが文脈自由文法を特徴付けている．

プッシュダウンオートマトン（スタックを持つ有限オートマトン）

文法の例　-- a**n・b**n (a**n は a の n 個連結を表す)

N = { S }

T = { a, b }

P = { S -> aSb, S -> ab }

開始記号 : S

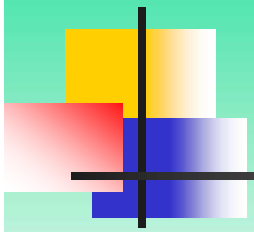
正規文法 (regular grammar)

生成規則に
　　A -> a または A -> aB　(A -> Ba)
という制約を設けた文法である．
ただしＡとＢは非終端記号であり，a は終端記号である．

左辺は１個の非終端記号でなければならず，右辺も限定されたパターンのみ
許されることが，正規文法を特徴付けている．

正規言語を受理する仮想機械（有限オートマトン）には現在の状態以外の記憶がない．

文法の例　-- a**m・b**n (a**n は a の n 個連結を表す)
```
N = { S, A }
T = { a, b }
P = { S -> aS, S -> aA, A -> bA, A -> b }
開始記号 : S
```



関数型言語とラムダ計算の話

手続き型言語

- 抽象化レベルが低い
  - ハードウェアの効率的な利用が可能
  - 記述能力が低い (how)
- 変数の書き換え
- フローチャートで表現できるグラフ

関数型言語

- 抽象化レベルが高い
  - ハードウェアの効率的な利用は困難
  - 記述能力が高い (what)
- 副作用がない
- ラムダ計算

型なしラムダ計算

ラムダ式とは
　変数 x, y, … | ラムダ抽象 λx.M | 関数適用 MN
である。ここで x,y,… は変数 M,N,… はラムダ式とする。

ラムダ計算とはβ-redexを見つけて簡約することである。
ここでβ-redexとは
　(λx.M)N
を指す。

関数型言語でよく使われる無名関数、高階関数、遅延評価などの理解に役立つ。

ハードウェアの話では論理値や整数を基本的な要素と考えるのが自然。
しかしラムダ計算では基本的な要素はラムダ式のみ。

ラムダ式の表現能力

ラムダ式がチューリング完全(Turing-Complete)であることの概要説明

いろいろなデータをラムダ式だけで表現可能
条件式を表現可能
Y-Combinator(不動点演算子) → 再帰関数定義可能 → 繰り返し

論理値

true	λtf.t	
false	λtf.f	
if	λpxy.pxy	
and	λpq.pq<false>	<false>はλtf.fに置き換えて計算することを意味する。
or	λpq.p<true>q	一般的に使われる記号ではない。
not	λp.p<false><true>	

整数

0	λsz.z
1	λsz.sz
2	λsz.s(sz)
3	λsz.s(s(sz))
succ	λnsz.s(nsz)
plus	λmnsz.ms(nsz)

再帰関数

Y	λf.(λx.f(xx))(λx.f(xx))
YTuring	(λxf.f(xxf))(λxf.f(xxf))

ラムダ式の加算

記号 ＝ をラムダ計算とは無関係な文字列定義 <name> を文字列置換と解釈する。

plus ＝ λmnsz.ms(nsz)
2 ＝ λsz.s(sz)
3 ＝ λsz.s(s(sz))
5 ＝ λsz.s(s(s(sz)))

とすると次のような加算結果が得られる。

<plus><3><2> → <5>

<plus><3><2> <xxx> は対応するラムダ式に置換
＝ (λmnsz.ms(nsz))(λsz.s(s(sz)))(λsz.s(sz))
→ (λnsz.(λsz.s(s(sz)))s(nsz))(λsz.s(sz))
→ λsz.(λsz.s(s(sz)))s((λsz.s(sz))sz)
→ λsz.(λz.s(s(sz)))((λsz.s(sz))sz)
→ λsz.s(s(s((λsz.s(sz))sz)))
→ λsz.s(s(s((λz.s(sz))z)))
→ λsz.s(s(s(sz)))

不動点演算子を使って4の階乗を計算する例

純粋なラムダ計算のみで条件分岐と繰り返し（再帰）を行っていることに注意
　ここでは記号 ＝ をラムダ計算とは無関係な文字列定義 <name> を文字列置換と解釈している。

```
if    = λpxy.pxy                          iszero = λn.n(λx.(λtf.f))(λtf.t)
mult  = λmns.m(ns)                        pred   = λnsz.n(λgh.h(gs))(λu.z)(λu.u)
1     = λsz.sz                            4      = λsz.s(s(s(sz)))
Y     = λf.(λx.f(xx))(λx.f(xx))
fact  = (λfx.<if>(<iszero>x)<1>(<mult>x(f(<pred>x))))     <xxx> は対応するラムダ式に置換
```

とすると４の階乗は次のようになる。実際の計算過程は複雑過ぎるので省略。

```
<Y><fact><4>
= (
    (λf.(λx.f(xx))(λx.f(xx)))                          -- <Y>
    (
      λfx.(λpxy.pxy)                                   -- λfx.<if>
        ((λn.n(λx.(λtf.f))(λtf.t))x)                   -- <iszero>x
        (λsz.sz)                                       -- <1>
        (
          (λmns.m(ns))                                 -- <mult>
            x                                          -- x
            (f((λnsz.n(λgh.h(gs))(λu.z)(λu.u))x))      -- f(<pred>x)
        )
    )
  )
  (λsz.s(s(s(sz))))                                    -- <4>
→ λsz.s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(sz))))))))))))))))))))))))
4068 step(s) [9.960ms]     この所要時間は参考値でありシステムに依存する。
```

部分適用 (高階関数の応用)

ラムダ計算と高階関数の自然な結び付きを示す例として**部分適用**を取り上げる.
部分適用の実例として
　　(λmn.m+n)3 → λn.3+n
を示す. ただし + や 3 など略記法を使っていることに注意.
意味は
　　２変数関数 λmn.m+n を引数３に**部分適用**すると１変数関数 λn.3+n になる
と表現できる.

```
   (λmn.m+n)3      略記法
= <plus><3>        <xxx> は対応するラムダ式
= (λmnsz.ms(nsz))(λsz.s(s(sz)))
→ λnsz.(λsz.s(s(sz)))s(nsz)
→ λnsz.(λz.s(s(sz)))(nsz)
→ λnsz.s(s(s(nsz)))
= λn.3+n      略記法
```

この結果が引数に３を加算する関数となっていることを確認するために結果の関数を２に適用して
　　(λn.3+n)2 → 5
となることを示す.

```
   (λn.3+n)2      略記法
= (λnsz.s(s(s(nsz))))(λsz.s(sz))
→ λsz.s(s(s((λsz.s(sz))sz)))
→ λsz.s(s(s((λz.s(sz))z)))
→ λsz.s(s(s(s(sz))))
= 5      略記法
```

関数合成（高階関数の応用）

ラムダ計算と高階関数の自然な結び付きを示す第２の例として関数合成を取り上げる．
関数 f と g から合成関数を返す関数は λfgx.g(fx) で表現できる．
関数合成の実例として
　　(λfgx.g(fx))(λx.1+x)(λx.2*x) → λx.2*(1+x)
を示す．ただし +,* や 1,2 など略記法を使っていることに注意．
意味は
　　１を加算する関数と２倍する関数の合成関数は１を加算した結果を２倍する関数
と表現できる．

　(λfgx.g(fx))(λx.1+x)(λx.2*x)　　略記法
＝ (λfgx.g(fx))(<plus><1>)(<mult><2>)　　<xxx> は対応するラムダ式
＝ (λfgx.g(fx))((λmnsz.ms(nsz))(λsz.sz))((λmns.m(ns))(λsz.s(sz)))
→ (λgx.g((λmnsz.ms(nsz))λsz.sz)x))((λmns.m(ns))λsz.s(sz))
→ λx.((λmns.m(ns))λsz.s(sz))(((λmnsz.ms(nsz))λsz.sz)x)
　　(省略)
→ λxsz.s(xs((λz.sz)(xsz)))
→ λxsz.s(xs(s(xsz)))
＝ λx.2*(1+x)　　略記法

実際にこの結果が前述の合成関数となっていることを確認するために結果の関数を５に適用して
　　(λx.2*(1+x))5 → 12
となることを示す．

　(λx.2*(1+x))5　　略記法
＝ (λxsz.s(xs(s(xsz))))(λsz.s(s(s(s(sz)))))
→ λsz.s((λsz.s(s(s(s(sz)))))s(s((λsz.s(s(s(s(sz)))))sz)))
→ λsz.s((λz.s(s(s(s(sz)))))(s((λsz.s(s(s(s(sz)))))sz)))
→ λsz.s(s(s(s(s(s(s((λsz.s(s(s(s(sz)))))sz)))))))
→ λsz.s(s(s(s(s(s(s((λz.s(s(s(s(sz)))))z)))))))
→ λsz.s(s(s(s(s(s(s(s(s(s(sz)))))))))))
＝ 12　　略記法

代入操作

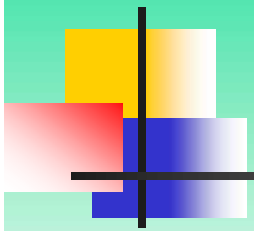
代入操作の定義（プログラミング言語の基礎理論 大堀淳著 p.18)

	形式	条件	結果	
(a)	x[x:=N]		N	
(b)	y[x:=N]	x≠y	y	
(c)	(λy.M)[x:=N]	x=y	λy.M	
(d)	(λy.M)[x:=N]	x≠y y !∈ FV(N)	λy.M[x:=N]	[注1]
(e)	(λy.M)[x:=N]	上記以外	λz.M[y:=z][x:=N]	[注2]
(f)	(M1M2)[x:=N]		M1[x:=N]M2[x:=N]	

[注1] y !∈ FV(N) は y が N の自由変数ではないことを意味する.
[注2] 自動的にα変換されることに注意. 変数 z はまったく新しい名前が選ばれる.

計算例

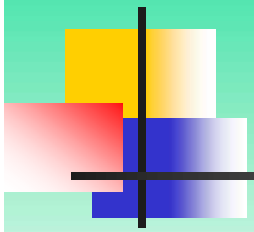
(λy.xy)[y:=z] → λy.xy (c)の場合
(λy.xy)[x:=z] → λy.zy (d)の場合
(λy.xy)[x:=yz] → λx.yzx (e)の場合



β変換

β変換の定義（計算論 高橋正子著 p.65）
記号→はβ変換を意味するものとする．

規則	例
(A) $(\lambda x.M)N \rightarrow M[x:=N]$	$(\lambda x.x)M \rightarrow M$ $(\lambda x.y)M \rightarrow y$ $(\lambda x.\lambda y.x)z \rightarrow \lambda y.z$
(B) M → N ならば $\lambda x.M \rightarrow \lambda x.N$	$\lambda x.(\lambda y.y)x \rightarrow \lambda x.x$
(C) M → N ならば MP → NP	$(\lambda a.b)cd \rightarrow bd$
(D) M → N ならば PM → PN	$x((\lambda x.y)z) \rightarrow xy$



最左簡約の優先

複数の簡約結果がある場合は最左簡約を採用する．

最左簡約が停止しないラムダ式はどれを選んでも停止しない．
もし正規形を持つならばどれを選んでも同じ正規形となる．

計算例

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

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