ものまね鳥を愛でる

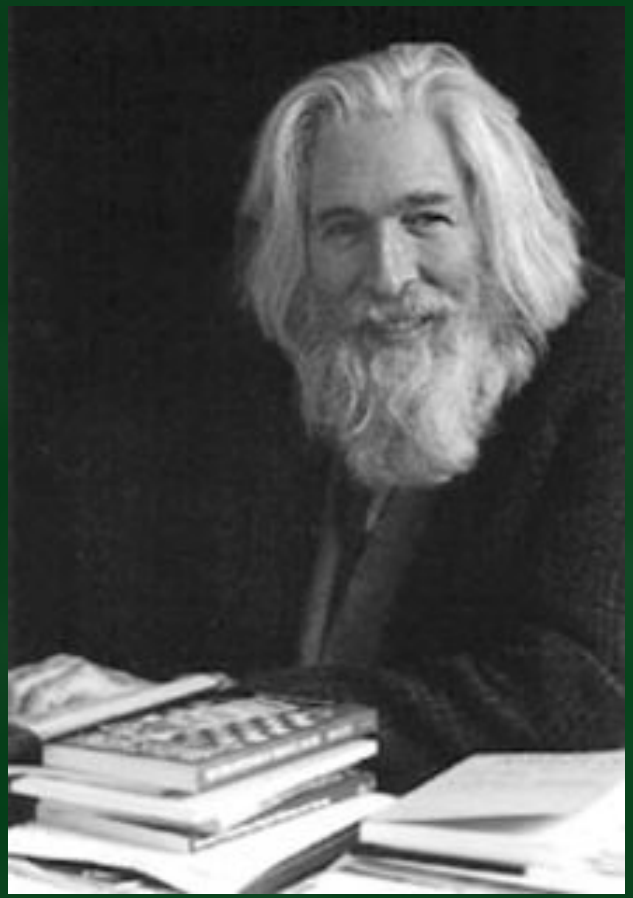
結合子論理と計算

石井大海

筑波大学博士後期課程一年

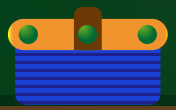
Monday 6th March, 2017

追悼



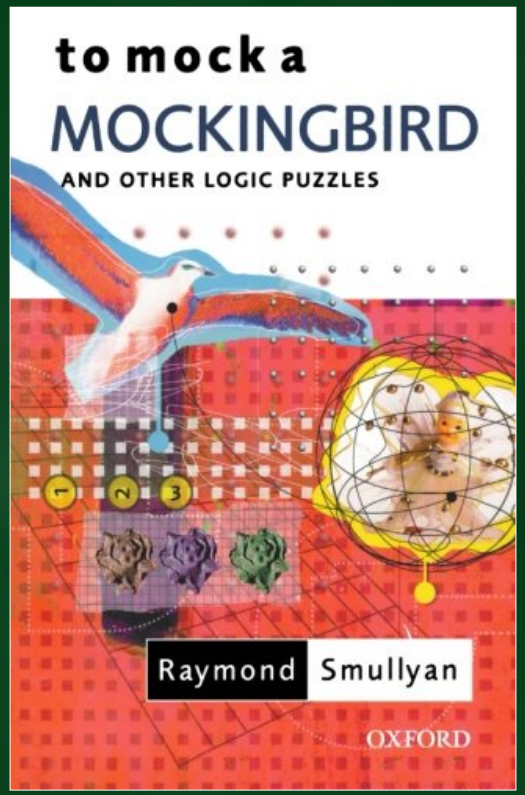
Raymond M. Smullyan,
(1919-2017)

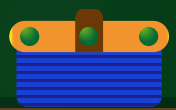
Table Of Contents



ものまね鳥をまねる

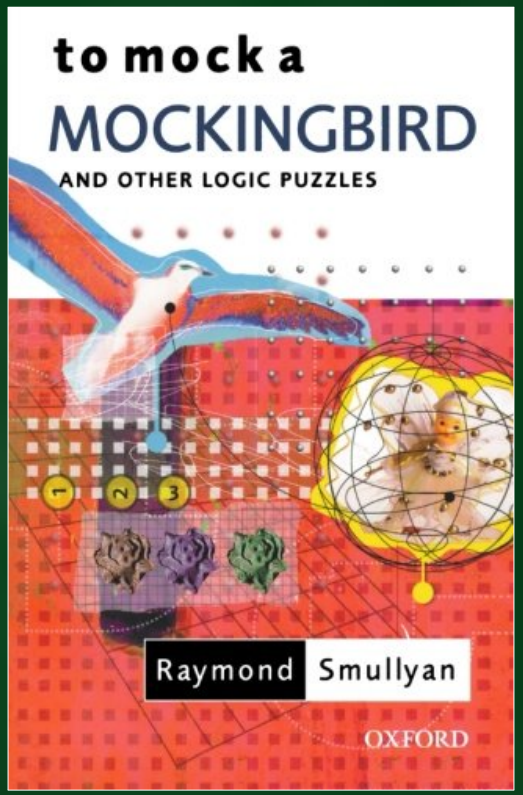
- 今回の種本

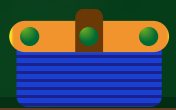




ものまね鳥をまねる

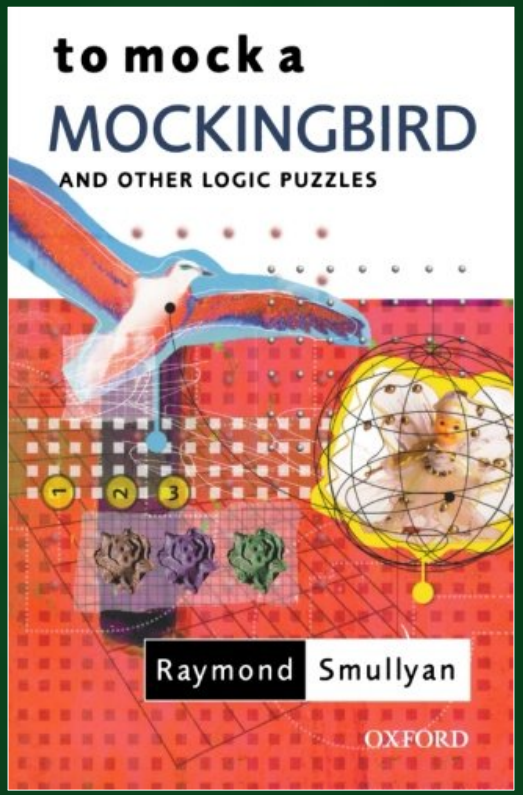
- 今回の種本
  - 訳書 [5] は余り良くないので原書の方が良いかも





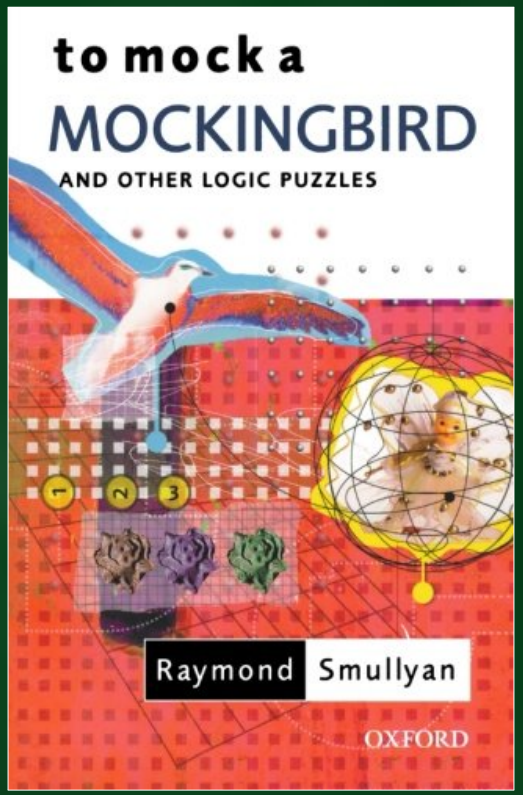
ものまね鳥をまねる

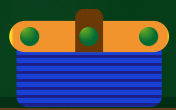
- 今回の種本
  - 訳書 [5] は余り良くないので原書の方が良いかも
- 結合子を「鳥」に見立てたパズルに親しむうちに不完全性定理に行き着く.



ものまね鳥をまねる

- 今回の種本
  - 訳書 [5] は余り良くないので原書の方が良いかも
- 結合子を「鳥」に見立てたパズルに親しむうちに不完全性定理に行き着く.





鳥のパズル（設定）

種本の冒頭では次の設定でパズルを考えている：

- ある森に棲む鳥は，ある鳥の名前を聞くと別の鳥の名前を言う.

鳥のパズル（設定）

種本の冒頭では次の設定でパズルを考えている：

- ある森に棲む鳥は，ある鳥の名前を聞くと別の鳥の名前を言う．
  - 鳥 A の「B」に対する反応を AB と書く．括弧は左結合と考えて省略する：$ABC = (AB)C$.

鳥のパズル（設定）

種本の冒頭では次の設定でパズルを考えている：

- ある森に棲む鳥は，ある鳥の名前を聞くと別の鳥の名前を言う．
  - 鳥 A の「B」に対する反応を AB と書く．括弧は左結合と考えて省略する：$ABC = (AB)C$.
- この森にはこういう鳥がいる：

鳥のパズル（設定）

種本の冒頭では次の設定でパズルを考えている：

- ある森に棲む鳥は，ある鳥の名前を聞くと別の鳥の名前を言う.
  - 鳥 A の「B」に対する反応を AB と書く．括弧は左結合と考えて省略する：$ABC = (AB)C$.
- この森にはこういう鳥がいる：
  1. （合成鳥）任意の鳥 X と Y に対して，そいつらをリレーした結果を返すような合成鳥 Z が必ずいる.

$$\text{i.e. } \forall X\, \forall Y\, \exists Z\, \forall x\, Zx = X(Yx).$$

X, Y に対しその合成鳥（の一つ）を $\mathbf{B}XY$ で表す.

鳥のパズル（設定）

種本の冒頭では次の設定でパズルを考えている：

- ある森に棲む鳥は，ある鳥の名前を聞くと別の鳥の名前を言う．
  - 鳥 A の「B」に対する反応を AB と書く．括弧は左結合と考えて省略する：$ABC = (AB)C$.
- この森にはこういう鳥がいる：
  1. （合成鳥）任意の鳥 X と Y に対して，そいつらをリレーした結果を返すような合成鳥 Z が必ずいる．

     $$\text{i.e. } \forall X\, \forall Y\, \exists Z\, \forall x\, Zx = X(Yx).$$

     X, Y に対しその合成鳥（の一つ）を $\mathbf{B}XY$ で表す．
  2. （物まね鳥）$\mathbf{M}x = xx$ を満たすような物まね鳥 $\mathbf{M}$ が居る．

鳥のパズル（設定）

種本の冒頭では次の設定でパズルを考えている：

- ある森に棲む鳥は，ある鳥の名前を聞くと別の鳥の名前を言う．
  - 鳥 A の「B」に対する反応を AB と書く．括弧は左結合と考えて省略する：$ABC = (AB)C$.
- この森にはこういう鳥がいる：
  1. （合成鳥）任意の鳥 X と Y に対して，そいつらをリレーした結果を返すような合成鳥 Z が必ずいる．

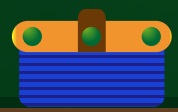
     $$\text{i.e. } \forall X\, \forall Y\, \exists Z\, \forall x\, Zx = X(Yx).$$

     X, Y に対しその合成鳥（の一つ）を $\mathbf{B}XY$ で表す．
  2. （物まね鳥）$\mathbf{M}x = xx$ を満たすような物まね鳥 $\mathbf{M}$ が居る．
- $AB = B$ となるとき，「鳥 A は B が好き」と言う．

例

問 1 (不動点定理)

合成鳥 B と物まね鳥 M が居るなら，どんな鳥 A も少なくとも一羽の鳥が好き (i.e. $\forall A\, \exists x\, Ax = x$).



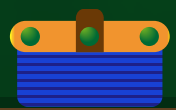
例

問 1 (不動点定理)

合成鳥 **B** と物まね鳥 **M** が居るなら，どんな鳥 A も少なくとも一羽の鳥が好き (i.e. $\forall A\, \exists x\, Ax = x$).

(答) $X = \mathbf{B}A\mathbf{M}$ とすると XX がそれ.

$$XX = A(MX) = A(XX)$$



例

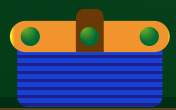
問 1 (不動点定理)

合成鳥 **B** と物まね鳥 **M** が居るなら，どんな鳥 A も少なくとも一羽の鳥が好き (i.e. $\forall A\, \exists x\, Ax = x$).

(答) $X = \mathbf{B}A\mathbf{M}$ とすると XX がそれ.

$$XX = A(MX) = A(XX)$$

- こんな感じで帳尻を合わせていくパズルが沢山.



例

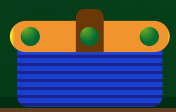
問 1 (不動点定理)

合成鳥 **B** と物まね鳥 **M** が居るなら，どんな鳥 A も少なくとも一羽の鳥が好き (i.e. $\forall A\, \exists x\, Ax = x$).

(答) $X = \mathbf{B}A\mathbf{M}$ とすると XX がそれ.

$$XX = A(MX) = A(XX)$$

- こんな感じで帳尻を合わせていくパズルが沢山.
- 詳しくは紹介しませんが，次のパズルは言霊がすごい：



例

問 1 (不動点定理)

合成鳥 **B** と物まね鳥 **M** が居るなら，どんな鳥 A も少なくとも一羽の鳥が好き (i.e. $\forall A\, \exists x\, Ax = x$).

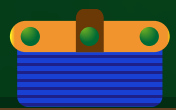
(答) $X = \mathbf{B}A\mathbf{M}$ とすると XX がそれ.

$$XX = A(MX) = A(XX)$$

- こんな感じで帳尻を合わせていくパズルが沢山.
- 詳しくは紹介しませんが，次のパズルは言霊がすごい：

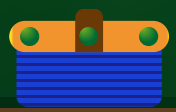
問 2

救いがたいほど自己中心的なヒバリは異常に魅力的なのはなぜか？



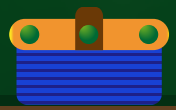
数学的内容：結合子論理

? 「鳥」パズルは数学的には何なのか？



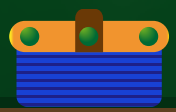
数学的内容：結合子論理

- ? 「鳥」パズルは数学的には何なのか？
  - ⇝ 結合子論理！



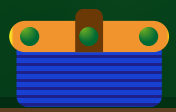
数学的内容：結合子論理

- ? 「鳥」パズルは数学的には何なのか？
  - ⇝ 結合子論理！
- 結合子と変数の文字列を一定の規則に従って書き換えていく「計算」だと思える.



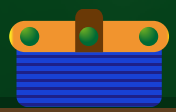
数学的内容：結合子論理

- ? 「鳥」パズルは数学的には何なのか？
  - ⇝ 結合子論理！
- 結合子と変数の文字列を一定の規則に従って書き換えていく「計算」だと思える.
  - ※ ここでは括弧の順番まで込めた文字列を考えている. また, 三つ以上の文字が並ぶ時は左から順に二つずつ括弧が省略されていると考える：$xyz = (xy)z = ((xy)z)$, $x(yzw) = (x((yz)w))$.



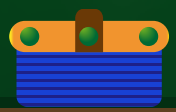
数学的内容：結合子論理

- ? 「鳥」パズルは数学的には何なのか？
  - ⇝ 結合子論理！
- 結合子と変数の文字列を一定の規則に従って書き換えていく「計算」だと思える.
  - ※ ここでは括弧の順番まで込めた文字列を考えている. また, 三つ以上の文字が並ぶ時は左から順に二つずつ括弧が省略されていると考える：$xyz = (xy)z = ((xy)z)$, $x(yzw) = (x((yz)w))$.
- 先程の問題は, 「$\mathbf{B}xyz \triangleright x(yz)$, $\mathbf{M}x \triangleright xx$ という書き換え規則の下で, 任意の A に対してその不動点として振る舞うものを見付けよ」という意味だった.



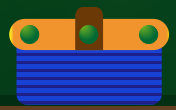
数学的内容：結合子論理

- ? 「鳥」パズルは数学的には何なのか？
  - ⇝ 結合子論理！
- 結合子と変数の文字列を一定の規則に従って書き換えていく「計算」だと思える.
  - ※ ここでは括弧の順番まで込めた文字列を考えている. また, 三つ以上の文字が並ぶ時は左から順に二つずつ括弧が省略されていると考える：$xyz = (xy)z = ((xy)z)$, $x(yzw) = (x((yz)w))$.
- 先程の問題は,「$\mathbf{B}xyz \triangleright x(yz)$, $\mathbf{M}x \triangleright xx$ という書き換え規則の下で, 任意の A に対してその不動点として振る舞うものを見付けよ」という意味だった.
- ? 自由に「書き換え」出来るにはどんな結合子があればよいか？



数学的内容：結合子論理

- ⑦ 「鳥」パズルは数学的には何なのか？
  - ⇝ 結合子論理！
- 結合子と変数の文字列を一定の規則に従って書き換えていく「計算」だと思える.
  - ※ ここでは括弧の順番まで込めた文字列を考えている. また, 三つ以上の文字が並ぶ時は左から順に二つずつ括弧が省略されていると考える：$xyz = (xy)z = ((xy)z)$, $x(yzw) = (x((yz)w))$.
- 先程の問題は,「$\mathbf{B}xyz \triangleright x(yz)$, $\mathbf{M}x \triangleright xx$ という書き換え規則の下で, 任意の A に対してその不動点として振る舞うものを見付けよ」という意味だった.
- ⑦ 自由に「書き換え」出来るにはどんな結合子があればよいか？
- ⑦ 「書き換え」計算でどの程度の事が計算出来るのか（計算能力）？



自由に「書き換え」が出来るために

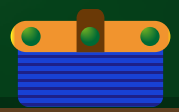
- 各「書き換え」ステップの定義が必要.

自由に「書き換え」が出来るために

- 各「書き換え」ステップの定義が必要.
  - ⇝ ここではラフに「変数 $x_1 \dots x_n$ をとって，それらと幾つかの結合子からなる括弧つき文字列を返す操作」と思っておこう.

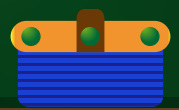
自由に「書き換え」が出来るために

- 各「書き換え」ステップの定義が必要.
  - ⇝ ここではラフに「変数 $x_1 \ldots x_n$ をとって，それらと幾つかの結合子からなる括弧つき文字列を返す操作」と思っておこう.
- 変数 x を含む文字列 t に対して，「x を t に書き換える操作」を "$\lambda\, x.\, t$" と表すことにする.



自由に「書き換え」が出来るために

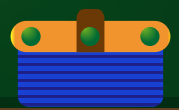
- 各「書き換え」ステップの定義が必要.
  - ⇝ ここではラフに「変数 $x_1 \dots x_n$ をとって，それらと幾つかの結合子からなる括弧つき文字列を返す操作」と思っておこう.
- 変数 x を含む文字列 t に対して，「x を t に書き換える操作」を "$\lambda\, x.\, t$" と表すことにする.
  - ◎ $(\lambda\, x.\, t)A \rhd t\left[\begin{smallmatrix} x \\ A \end{smallmatrix}\right]$.（$t\left[\begin{smallmatrix} x \\ A \end{smallmatrix}\right]$ は t で変数 x に A を代入したもの）



自由に「書き換え」が出来るために

- 各「書き換え」ステップの定義が必要.
  - ⇝ ここではラフに「変数 $x_1 \ldots x_n$ をとって，それらと幾つかの結合子からなる括弧つき文字列を返す操作」と思っておこう.
- 変数 x を含む文字列 t に対して，「x を t に書き換える操作」を "$\lambda\, x.\ t$" と表すことにする.
  - ◎ $(\lambda\, x.\ t)A \rhd t[{}^{x}_{A}]$. ($t[{}^{x}_{A}]$ は t で変数 x に A を代入したもの)
- 複数変数を必要とする場合は

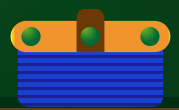
  $\lambda\, x_1 \ldots x_n.\ t \equiv \lambda\, x_1.\ \lambda\, x_2.\ \ldots \lambda\, x_n.\ t$ と表す.



自由に「書き換え」が出来るために

- 各「書き換え」ステップの定義が必要.
  - ⇝ ここではラフに「変数 $x_1 \ldots x_n$ をとって，それらと幾つかの結合子からなる括弧つき文字列を返す操作」と思っておこう.
- 変数 x を含む文字列 t に対して，「x を t に書き換える操作」を "$\lambda\, x.\, t$" と表すことにする.
  - ◎ $(\lambda\, x.\, t)A \rhd t[{}^{x}_{A}]$. ($t[{}^{x}_{A}]$ は t で変数 x に A を代入したもの)
- 複数変数を必要とする場合は

  $\lambda\, x_1 \ldots x_n.\, t \equiv \lambda\, x_1.\ \lambda\, x_2.\ \ldots \lambda\, x_n.\, t$ と表す.
- この問題は次のように言い換えられる：



自由に「書き換え」が出来るために

- 各「書き換え」ステップの定義が必要.
  - $\rightsquigarrow$ ここではラフに「変数 $x_1 \ldots x_n$ をとって，それらと幾つかの結合子からなる括弧つき文字列を返す操作」と思っておこう.
- 変数 x を含む文字列 t に対して，「x を t に書き換える操作」を "$\lambda x.\ t$" と表すことにする.
  - ◎ $(\lambda x.\ t)A \rhd t[{}^{x}_{A}]$. ($t[{}^{x}_{A}]$ は t で変数 x に A を代入したもの)
- 複数変数を必要とする場合は

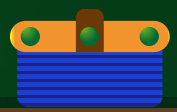
  $\lambda x_1 \ldots x_n.\ t \equiv \lambda x_1.\ \lambda x_2.\ \ldots \lambda x_n.\ t$ と表す.
- この問題は次のように言い換えられる：

問 3

$\vec{x}$ と t に対して，$(\lambda \vec{x}.\ t)\vec{x}$ と $C\vec{x}$ の簡約結果が一致する文字列 C が欲しい．どんな結合子があれば十分か？

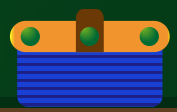
SK一計算：必要な結合子の見付け方

- 今回考える「文字列」t は変数を含む「文字」からはじめて二つずつ括弧で括ったものだった.



SK−計算：必要な結合子の見付け方

- 今回考える「文字列」t は変数を含む「文字」からはじめて二つずつ括弧で括ったものだった.
- ⇝ この作り方にそって考えてみよう！

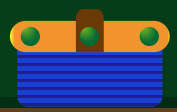


SK-計算：必要な結合子の見付け方

- 今回考える「文字列」t は変数を含む「文字」からはじめて二つずつ括弧で括ったものだった.

$\rightsquigarrow$ この作り方にそって考えてみよう！

1. $t = x$ の時：$(\lambda\, x.\, x)$ に当たるような結合子が欲しい. そこで $\mathsf{I} \equiv (\lambda\, x.\, x)$ とおこう.

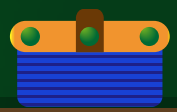


SK-計算：必要な結合子の見付け方

- 今回考える「文字列」t は変数を含む「文字」からはじめて二つずつ括弧で括ったものだった.

$\rightsquigarrow$ この作り方にそって考えてみよう！

1. $t = x$ の時：$(\lambda\, x.\, x)$ に当たるような結合子が欲しい．そこで $\mathbf{I} \equiv (\lambda\, x.\, x)$ とおこう.
2. t が x と異なる文字 y の時：$(\lambda\, x.\, y)$ は y を返す定数関数．そこで $\mathbf{K} \equiv (\lambda\, yx.\, y)$ とすれば，$\lambda\, x.\, y \equiv \mathbf{K}y$ と書ける.

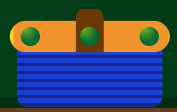


SK-計算：必要な結合子の見付け方

- 今回考える「文字列」t は変数を含む「文字」からはじめて二つずつ括弧で括ったものだった.

$\rightsquigarrow$ この作り方にそって考えてみよう！

1. $t = x$ の時：$(\lambda\, x.\, x)$ に当たるような結合子が欲しい. そこで $\mathbf{I} \equiv (\lambda\, x.\, x)$ とおこう.
2. t が x と異なる文字 y の時：$(\lambda\, x.\, y)$ は y を返す定数関数. そこで $\mathbf{K} \equiv (\lambda\, yx.\, y)$ とすれば, $\lambda\, x.\, y \equiv \mathbf{K}y$ と書ける.
3. $t = (uv)$ の時：帰納法の仮定から, $(\lambda\, x.\, u)$ と $(\lambda\, x.\, v)$ は何らかの文字列で表せるとしてよい. このとき $uv \equiv ((\lambda\, x.\, u)x)((\lambda\, x.\, v)x)$ となるので, $\mathbf{S} \equiv \lambda\, xyz.\, (xz)(yz)$ とおけば, $\lambda\, x.\, uv \equiv S(\lambda\, x.\, u)(\lambda\, x.\, v)$.



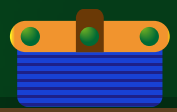
SK-計算：必要な結合子の見付け方

- 今回考える「文字列」t は変数を含む「文字」からはじめて二つずつ括弧で括ったものだった.

$\rightsquigarrow$ この作り方にそって考えてみよう！

1. $t = x$ の時：$(\lambda\, x.\, x)$ に当たるような結合子が欲しい. そこで $\mathbf{I} \equiv (\lambda\, x.\, x)$ とおこう.
2. t が x と異なる文字 y の時：$(\lambda\, x.\, y)$ は y を返す定数関数. そこで $\mathbf{K} \equiv (\lambda\, yx.\, y)$ とすれば, $\lambda\, x.\, y \equiv \mathbf{K}y$ と書ける.
3. $t = (uv)$ の時：帰納法の仮定から, $(\lambda\, x.\, u)$ と $(\lambda\, x.\, v)$ は何らかの文字列で表せるとしてよい. このとき $uv \equiv ((\lambda\, x.\, u)x)((\lambda\, x.\, v)x)$ となるので, $\mathbf{S} \equiv \lambda\, xyz.\, (xz)(yz)$ とおけば, $\lambda\, x.\, uv \equiv S(\lambda\, x.\, u)(\lambda\, x.\, v)$.

$\rightsquigarrow$ 以上から上記のような $\mathbf{S}, \mathbf{K}, \mathbf{I}$ があれば十分！



SK-計算：必要な結合子の見付け方

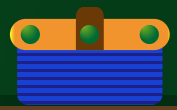
- 今回考える「文字列」t は変数を含む「文字」からはじめて二つずつ括弧で括ったものだった.

$\rightsquigarrow$ この作り方にそって考えてみよう！

1. $t = x$ の時：$(\lambda\, x.\, x)$ に当たるような結合子が欲しい. そこで $\mathbf{I} \equiv (\lambda\, x.\, x)$ とおこう.
2. t が x と異なる文字 y の時：$(\lambda\, x.\, y)$ は y を返す定数関数. そこで $\mathbf{K} \equiv (\lambda\, yx.\, y)$ とすれば, $\lambda\, x.\, y \equiv \mathbf{K}y$ と書ける.
3. $t = (uv)$ の時：帰納法の仮定から, $(\lambda\, x.\, u)$ と $(\lambda\, x.\, v)$ は何らかの文字列で表せるとしてよい. このとき $uv \equiv ((\lambda\, x.\, u)x)((\lambda\, x.\, v)x)$ となるので, $\mathbf{S} \equiv \lambda\, xyz.\, (xz)(yz)$ とおけば, $\lambda\, x.\, uv \equiv S(\lambda\, x.\, u)(\lambda\, x.\, v)$.

$\rightsquigarrow$ 以上から上記のような $\mathbf{S}, \mathbf{K}, \mathbf{I}$ があれば十分！

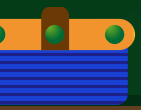
$\star$ 更に $\mathbf{I} \equiv \mathbf{SKK}$ と書ける（演習問題！）ので, $\mathbf{S}$ と $\mathbf{K}$ があれば書き換えは自由に出来る！



実際の例

これまでの議論で次の「**SK**-変換」規則が得られた：

$$\begin{aligned} \lambda x.\, x &:\equiv \mathbf{I} \equiv \mathbf{SKK}, \\ \lambda x.\, y &:\equiv \mathbf{K}y \qquad\qquad (\text{if } x \neq y), \\ \lambda x.\, (uv) &:\equiv \mathbf{S}(\lambda x.\, u)(\lambda x.\, v). \end{aligned}$$



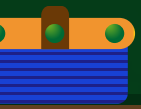
実際の例

これまでの議論で次の「**SK**-変換」規則が得られた：

$$\begin{aligned} \lambda x.\, x &:\equiv \mathbf{I} \equiv \mathbf{SKK}, \\ \lambda x.\, y &:\equiv \mathbf{K}y \qquad\qquad (\text{if } x \neq y), \\ \lambda x.\, (uv) &:\equiv \mathbf{S}(\lambda x.\, u)(\lambda x.\, v). \end{aligned}$$

Example 1

$$\mathbf{M} \equiv \lambda x.xx \equiv \mathbf{S}(\lambda x.\, x)(\lambda x.\, x) \equiv \mathbf{SII} \equiv \mathbf{S}(\mathbf{SKK})(\mathbf{SKK}).$$



実際の例

これまでの議論で次の「SK-変換」規則が得られた：

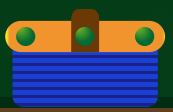
$$\begin{aligned}
\lambda x.\, x &:\equiv \mathbf{I} \equiv \mathbf{SKK}, \\
\lambda x.\, y &:\equiv \mathbf{K}y \qquad\qquad (\text{if } x \neq y), \\
\lambda x.\, (uv) &:\equiv \mathbf{S}(\lambda x.\, u)(\lambda x.\, v).
\end{aligned}$$

Example 1

$\mathbf{M} \equiv \lambda x.xx \equiv \mathbf{S}(\lambda x.\, x)(\lambda x.\, x) \equiv \mathbf{SII} \equiv \mathbf{S}(\mathbf{SKK})(\mathbf{SKK})$.

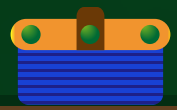
Exercise 1

上の SK-変換には結構無駄があるので，もう少し結果が短かくなる規則を与えよ．それを使って $\mathbf{B} \equiv \lambda xyz.x(yz)$ を S, K, I で表せ．



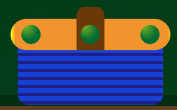
計算能力

- SK-計算でどの程度の事が計算出来るのか？



計算能力

- **SK**-計算でどの程度の事が計算出来るのか？
- ⇝ 実は理論上計算機と同じ計算が出来る！



計算能力

- SK-計算でどの程度の事が計算出来るのか？

⇝ 実は理論上計算機と同じ計算が出来る！

Theorem 2 (構造化定理)

計算機で計算可能なアルゴリズムを実装するには次の三つの要素があればよい：

逐次実行 `f; g` 「処理 f の後に g をやる」

条件分岐 `if (p) then f else g` 「条件 p が成り立つなら f, さもなくば g」

反復実行 `while (p) { f }` 「条件 p が成り立つ限り f を繰り返す」

計算能力

- **SK**-計算でどの程度の事が計算出来るのか？

⇝ 実は理論上計算機と同じ計算が出来る！

Theorem 2 (構造化定理)

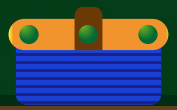
計算機で計算可能なアルゴリズムを実装するには次の三つの要素があればよい：

逐次実行 `f; g` 「処理 f の後に g をやる」

条件分岐 `if (p) then f else g` 「条件 p が成り立つなら f, さもなくば g」

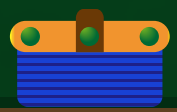
反復実行 `while (p) { f }` 「条件 p が成り立つ限り f を繰り返す」

⇝ 逐次実行は明らかなので，**SK**-計算で条件分岐と反復操作が出来ることがわかればよい！



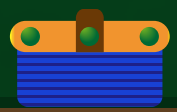
条件分岐と反復操作

? 条件分岐



条件分岐と反復操作

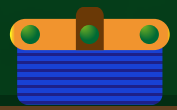
? 条件分岐
  ◎ 基本戦略：`if p then t else f` と書く代わりに $ptf(=(pt)f)$ と書こう.



条件分岐と反復操作

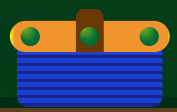
- ? 条件分岐
  - ◎ 基本戦略：`if p then t else f` と書く代わりに $ptf(= (pt)f)$ と書こう.

  $\rightsquigarrow$ $\mathtt{true} :\equiv \lambda\, t\, f.\, t \equiv \mathbf{K}$, $\mathtt{false} \equiv \lambda\, t\, f.\, f \equiv \mathbf{KI}$ とすれば良い.



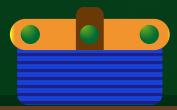
条件分岐と反復操作

- ? 条件分岐
  - ◎ 基本戦略：`if p then t else f` と書く代わりに $ptf(=(pt)f)$ と書こう.
  - $\rightsquigarrow$ `true` $:\equiv \lambda\, t\, f.\, t \equiv \mathbf{K}$, `false` $\equiv \lambda\, t\, f.\, f \equiv \mathbf{KI}$ とすれば良い.
- ? 反復操作……`while p f x` が「px が偽になるまで f を x に繰り返し適用」になるような `while` を得ればよい.



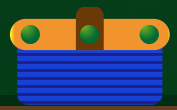
条件分岐と反復操作

- ? 条件分岐
  - ◎ 基本戦略：`if p then t else f` と書く代わりに $ptf\,(= (pt)f)$ と書こう.
  - $\rightsquigarrow$ `true` $:\equiv \lambda\, t\, f.\, t \equiv \mathbf{K}$, `false` $\equiv \lambda\, t\, f.\, f \equiv \mathbf{KI}$ とすれば良い.
- ? 反復操作……`while p f x` が「px が偽になるまで f を x に繰り返し適用」になるような `while` を得ればよい.
  - ◎ 観察：$\texttt{while}\, p\, f\, x = \texttt{if}\, (p\, x)\, \texttt{then}\, x\, \texttt{else}\, f\, (\texttt{while}\, p\, f\, x)$



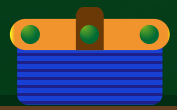
条件分岐と反復操作

- ? 条件分岐
  - ◎ 基本戦略：`if p then t else f` と書く代わりに $ptf\,(= (pt)f)$ と書こう.
  - ⇝ `true` $:\equiv \lambda\, t\, f.\, t \equiv \mathbf{K}$, `false` $\equiv \lambda\, t\, f.\, f \equiv \mathbf{KI}$ とすれば良い.
- ? 反復操作……`while p f x` が「px が偽になるまで f を x に繰り返し適用」になるような `while` を得ればよい.
  - ◎ 観察：$\texttt{while}\, p\, f\, x = \texttt{if}\,(p\,x)\,\texttt{then}\, x\,\texttt{else}\, f\,(\texttt{while}\, p\, f\, x)$
  - ⇝ $\lambda\, z.$ `if (p x) then x else f z` の不動点があればよい.



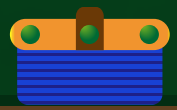
条件分岐と反復操作

- ? 条件分岐
  - ◎ 基本戦略：`if p then t else f` と書く代わりに $ptf\,(= (pt)f)$ と書こう.
  - ⇝ `true` $:\equiv \lambda\, t\, f.\, t \equiv \mathbf{K}$, `false` $\equiv \lambda\, t\, f.\, f \equiv \mathbf{KI}$ とすれば良い.
- ? 反復操作……`while p f x` が「px が偽になるまで f を x に繰り返し適用」になるような `while` を得ればよい.
  - ◎ 観察：$\texttt{while}\, p\, f\, x = \texttt{if}\,(p\,x)\,\texttt{then}\,x\,\texttt{else}\,f\,(\texttt{while}\, p\, f\, x)$
  - ⇝ $\lambda\, z.$ `if (p x) then x else f z` の不動点があればよい.
  - ⇝ **B**, **M** は **S**, **K** で書ける. すると，最初の問題から，**SK**-計算でも任意の項に対して不動点が存在する！



条件分岐と反復操作

- ? 条件分岐
  - ◎ 基本戦略：`if p then t else f` と書く代わりに $ptf\,(= (pt)f)$ と書こう.
  - ⇝ `true` $:\equiv \lambda\, t\, f.\, t \equiv \mathbf{K}$, `false` $\equiv \lambda\, t\, f.\, f \equiv \mathbf{KI}$ とすれば良い.
- ? 反復操作……`while p f x` が「px が偽になるまで f を x に繰り返し適用」になるような `while` を得ればよい.
  - ◎ 観察：$\texttt{while}\, p\, f\, x = \texttt{if}\,(p\,x)\,\texttt{then}\,x\,\texttt{else}\,f\,(\texttt{while}\, p\, f\, x)$
  - ⇝ $\lambda\, z.$ `if (p x) then x else f z` の不動点があればよい.
  - ⇝ **B**, **M** は **S**, **K** で書ける．すると，最初の問題から，**SK**-計算でも任意の項に対して不動点が存在する！
  - ◎ よって反復操作も実現可能！



条件分岐と反復操作

- ? 条件分岐
  - ◎ 基本戦略：`if p then t else f` と書く代わりに $ptf\,(= (pt)f)$ と書こう.
  - ⇝ `true` $:\equiv \lambda\, t\, f.\, t \equiv \mathbf{K}$, `false` $\equiv \lambda\, t\, f.\, f \equiv \mathbf{KI}$ とすれば良い.
- ? 反復操作……`while p f x` が「px が偽になるまで f を x に繰り返し適用」になるような `while` を得ればよい.
  - ◎ 観察：$\texttt{while}\, p\, f\, x = \texttt{if}\,(p\,x)\,\texttt{then}\,x\,\texttt{else}\,f\,(\texttt{while}\, p\, f\, x)$
  - ⇝ $\lambda\, z.$ `if (p x) then x else f z` の不動点があればよい.
  - ⇝ **B**, **M** は **S**, **K** で書ける. すると，最初の問題から，**SK**-計算でも任意の項に対して不動点が存在する！
  - ◎ よって反復操作も実現可能！
- ⋆ 構造化定理より **SK**-計算の能力は計算機と「等価」.

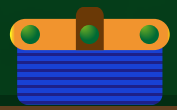
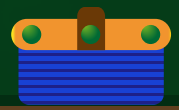


Table Of Contents



まとめとその他の話題

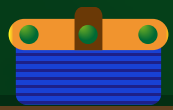
計算が出来るフレンズ

- **SK**-計算：$\mathbf{S}xyz \triangleright xz(yz)$, $\mathbf{K}xy \triangleright x$ なる二つの書き換え規則に従って式変形をする操作.

まとめとその他の話題

計算が出来るフレンズ

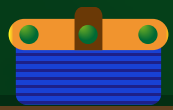
- **SK**-計算：$\mathbf{S}xyz \triangleright xz(yz)$, $\mathbf{K}xy \triangleright x$ なる二つの書き換え規則に従って式変形をする操作.
  - こういうのを結合子計算とか結合子論理という.



まとめとその他の話題

計算が出来るフレンズ

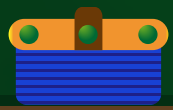
- **SK**-計算：**S***xyz* ▷ *xz*(*yz*), **K***xy* ▷ *x* なる二つの書き換え規則に従って式変形をする操作.
  - こういうのを結合子計算とか結合子論理という.
- ⋆ **SK** は計算機と同等の計算能力を持つ.



まとめとその他の話題

計算が出来るフレンズ

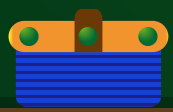
- **SK**-計算：**S***xyz* ▷ *xz*(*yz*), **K***xy* ▷ *x* なる二つの書き換え規則に従って式変形をする操作.
  - こういうのを結合子計算とか結合子論理という.
- ⋆ **SK** は計算機と同等の計算能力を持つ.
- 発展的話題



まとめとその他の話題

算数が出来るフレンズ

- **SK**-計算：$\mathbf{S}xyz \rhd xz(yz)$, $\mathbf{K}xy \rhd x$ なる二つの書き換え規則に従って式変形をする操作.
  - こういうのを結合子計算とか結合子論理という.
- ⋆ **SK** は計算機と同等の計算能力を持つ.
- 発展的話題
  - 自然数 $n \in \mathbb{N}$ と「n 回反復」関数 $\lambda fx.\overbrace{f(f(f\ldots(f}^{n}\,x)))$ と n を同一視して，自然数の算数を実現出来る（Church 符号化）



まとめとその他の話題

論理が出来るフレンズ

- **SK**-計算：$\mathbf{S}xyz \triangleright xz(yz)$, $\mathbf{K}xy \triangleright x$ なる二つの書き換え規則に従って式変形をする操作.
  - こういうのを結合子計算とか結合子論理という.
- ⋆ **SK** は計算機と同等の計算能力を持つ.
- 発展的話題
  - 自然数 $n \in \mathbb{N}$ と「n 回反復」関数 $\lambda fx.\overbrace{f(f(f\ldots(f}^{n}\,x)))$ と n を同一視して，自然数の算数を実現出来る（Church 符号化）
  - **S**, **K** を関数だと思って「型」をつけてみると，
    $\mathbf{S} : (\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma$, $\mathbf{K} : \alpha \to \beta \to \alpha$.

まとめとその他の話題

論理が出来るフレンズ

- **SK**-計算：$\mathbf{S}xyz \rhd xz(yz)$, $\mathbf{K}xy \rhd x$ なる二つの書き換え規則に従って式変形をする操作.
  - こういうのを結合子計算とか結合子論理という.
- ⋆ **SK** は計算機と同等の計算能力を持つ.
- 発展的話題
  - 自然数 $n \in \mathbb{N}$ と「n 回反復」関数 $\lambda fx.\overbrace{f(f(f\ldots(f}^{n}\,x)))$ と n を同一視して，自然数の算数を実現出来る（Church 符号化）
  - **S**, **K** を関数だと思って「型」をつけてみると，
    $\mathbf{S} : (\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma$, $\mathbf{K} : \alpha \to \beta \to \alpha$.
    - ⇝ これは通常の命題論理の公理から排中律を除いた公理と同じ！

まとめとその他の話題

論理が出来るフレンズ

- **SK**-計算：$\mathbf{S}xyz \rhd xz(yz)$, $\mathbf{K}xy \rhd x$ なる二つの書き換え規則に従って式変形をする操作.
  - こういうのを結合子計算とか結合子論理という.
- ⋆ **SK** は計算機と同等の計算能力を持つ.
- 発展的話題
  - 自然数 $n \in \mathbb{N}$ と「n 回反復」関数 $\lambda fx.\overbrace{f(f(f\ldots(f}^{n}\,x)))$ と n を同一視して，自然数の算数を実現出来る（Church 符号化）
  - **S**, **K** を関数だと思って「型」をつけてみると，
    $\mathbf{S} : (\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma$, $\mathbf{K} : \alpha \to \beta \to \alpha$.
    - ⇝ これは通常の命題論理の公理から排中律を除いた公理と同じ！
    - ● これに「例外処理」「短絡評価」を入れると普通の論理になる.

まとめとその他の話題

論理が出来るフレンズ

- **SK**-計算：$\mathbf{S}xyz \rhd xz(yz)$, $\mathbf{K}xy \rhd x$ なる二つの書き換え規則に従って式変形をする操作.
  - こういうのを結合子計算とか結合子論理という.
- ⋆ **SK** は計算機と同等の計算能力を持つ.
- 発展的話題
  - 自然数 $n \in \mathbb{N}$ と「n 回反復」関数 $\lambda fx.\overbrace{f(f(f\ldots(f}^{n}\,x)))$ と n を同一視して，自然数の算数を実現出来る（Church 符号化）
  - **S**, **K** を関数だと思って「型」をつけてみると，
    $\mathbf{S} : (\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma$, $\mathbf{K} : \alpha \to \beta \to \alpha$.
    - ⇝ これは通常の命題論理の公理から排中律を除いた公理と同じ！
    - ● これに「例外処理」「短絡評価」を入れると普通の論理になる.
    - ⓘ 論理と計算の深い関係を示唆している（Curry-Howard 対応）

参考文献 I

[1] J. Roger Hindley and J. P Seldin, **Lambda-calculus and combinators, an introduction**, Cambridge, UK: Cambridge University Press, 2008, ISBN: 9780521898850, URL: http://www.loc.gov/catdir/toc/ecip0810/2008006276.html.

[2] Jean-Louis Krivine, **Realizability algebras: a program to well order $\mathbb{R}$**, version 0, Logical Methods in Computer Science 7 (3:2 Aug. 10, 2011), DOI: 10.2168/LMCS-7(3:2)2011, arXiv: 1005.2395 [cs.LO].

[3] 古森雄一, **汎用システムとしての「ラムダ計算 + 論理」**, 2006, URL: http://www.math.s.chiba-u.ac.jp/~komori/symposium/suukaiken/komori8.pdf.

[4] 古森雄一 and 小野寛晰, **現代数理論理学序説**, 日本評論社, 2010.

[5] R・スマリヤン, **ものまね鳥をまねる**, trans. by 阿部剛久 et al., 原著：To Mock a Mockingbird, 森北出版, 1998.

[6] 魔法少女, **命題論理**, 2015.