カリー・ハワード同型対応入門

大西琢朗
takuro.onishi@gmail.com

哲学基礎文化学系ゼミナール II
2009 年 6 月 11 日

この講義で説明すること

カリー・ハワード同型対応って何？

- 簡単に言うと

  数学における証明の構造と，
  コンピュータ・プログラムの構造が，
  ぴったり一致する

  ということ．

先週の復習：証明の構造

$$\frac{\begin{array}{c}[A]\\ \mathcal{D}\\ B\end{array}}{A \to B} \qquad \frac{A \to B \quad A}{B}$$

- ならば導入則と除去則．

今日はプログラム

コンピュータは何をするか

- キーボードを叩くと文字を画面に表示する．
- DVD を挿入すると，映像を再生する．
- クリックすると，ソフトが立ち上がる．

- コンピュータの作業に共通の性質：

入力に応じて決まった出力を返す

関数概念の拡張

作業は一種の関数である

- 数学的な関数：例えば一次関数

$$y = 2x + 5$$

入力 $x = 3$ に対し，出力 $y = 11$ を返す．
入力 $x = 5$ に対し，出力 $y = 15$ を返す．
⋮

- 入力と出力として，数以外のものも考える．
関数概念の自然な拡張．

複合物としての作業

- 例えば Google 検索：
  - キーボードの入力を文字列データに変換する．
  - インターネットに接続し，その文字列を含んだ Web ページを探す．
  - 探し出したページを，リストにしてブラウザに表示する．
- 基本的な作業を組み合わせ，その手順を指定することで，複雑な作業をやらせる．
- コンピュータの作業に共通の性質：

段取りに従った複合的なプロセス

コンピュータとプログラム

- コンピュータの作業は一種の関数．
- コンピュータの作業は複合的なプロセス．

⇒ 関数への入力とその出力を組み合わせた，複合的なプロセス．

- プログラム＝そのプロセスの記述．
- コンピュータは，それを読んでその通りに実行する．

数式は一種のプログラム

$$(2 + 3) \times 5$$

- これは 25 という値を表わす．と同時に，
- 入力から出力への計算のプロセスを表わす．
  - 足し算関数に 2 と 3 を入力して，
  - その結果と 5 を掛け算関数に入力して，
  - 出力として計算結果 25 を得る．
- やってることは，コンピュータと同じ．
- 数式も一種のプログラムだ．

人はそれを「計算」と呼ぶ

コンピュータ＝計算機

計算とは

- 与えられた入力に対し，
- いろいろな基本的な作業（関数）を，
- 段取り通りの手順で実行していき，
- 有限時間内に出力を返す．

そんな作業，行為，プロセス．

コンピュータの作業は「計算」

プログラムとその構造

- プログラム＝計算プロセスの記述．
- 関数への入力・出力の組み合わせ．
- コンピュータはそれを読み，実行する．

プログラムの構造　＝　計算プロセスの構造

- プログラムを書く際に注意すべきことは何か？

エラー：型の不整合

「このファイルは *Windows Media Player* では再生できません…」的なエラー．

- **動画ファイル：入力**
- Media Player：**ファイルを入力として受け取り，動画を出力として返す関数．**
- **関数に対して，「不適切な入力」がなされたために生じたエラー．**
- **このようなエラーを「型の不整合」と言う．**

エラー：型の不整合

- このエラーは，ユーザーが引き起こしたもの．
- でも，関数の入力と出力を複雑に組み合わせていくと，計算プロセスのどこかで型の不整合が起こってしまうこともあるかもしれない．

型の不整合が，絶対に起こらないようなプログラムの書き方とは？

- プログラムのとるべき構造が見えてくる．

型付ラムダ計算

- 型の不整合を防ぐプログラムの書き方：

型付ラムダ計算
(typed lambda calculus)

というプログラミング言語を紹介する．
- 抽象的だとわかりにくいので，数式と組み合わせて説明する．

ちなみに

- プログラムについて研究するのは，計算機科学 (computer science)．
- しかし，型付ラムダ計算などの基本的な枠組についての研究は，広い意味での論理学に含まれる．

⇒ カリー・ハワード同型対応は，論理学の中のかなり性格を異にする分野の間での対応．

型（タイプ）

型＝データの種類

- コンピュータが扱うデータはすべて固有の「型（タイプ）」を持っている．
- データ：プログラムが表わすもの．

型（タイプ）

- プログラム：関数への入力・出力の組み合わせ．

⇒ データの種類：入力になるものと，入力される関数．

- 入力と関数の型がうまくかみ合っていないと，不整合．

関数を表わすプログラム

- 関数を表わすプログラムをどのように書くか，考えてみよう．
- 数学では，関数を変数を使って表わす．

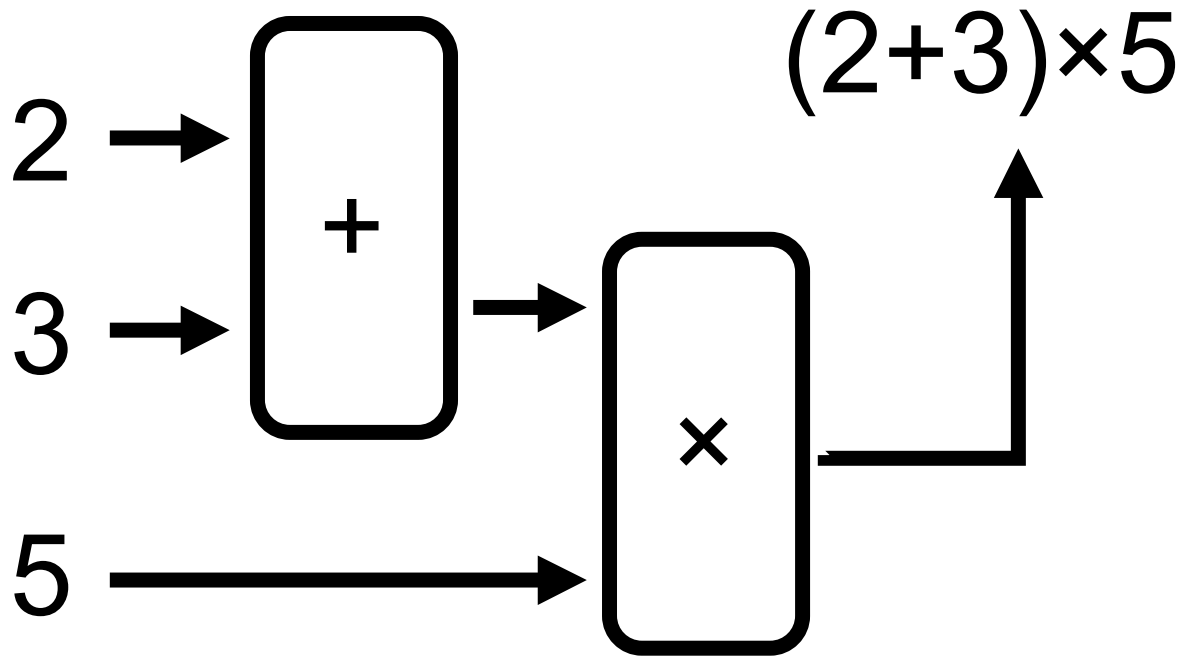
$$f(x) = (x + 3) \times 5$$

- でも実際のところ，「$(x + 3) \times 5$」は何を表わしているんだろうか？

変数のない数式

$$(2 + 3) \times 5$$

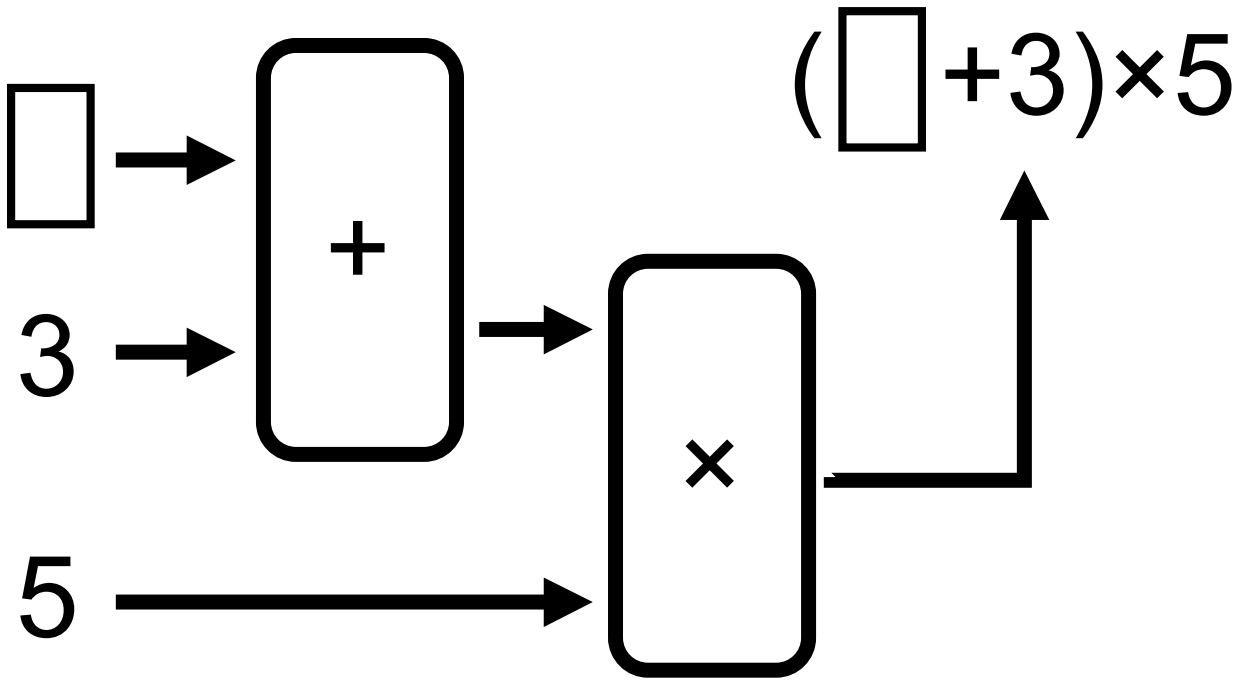
- 計算プロセスとその結果，両方を表わす．



変数を含む数式

$$(x + 3) \times 5$$

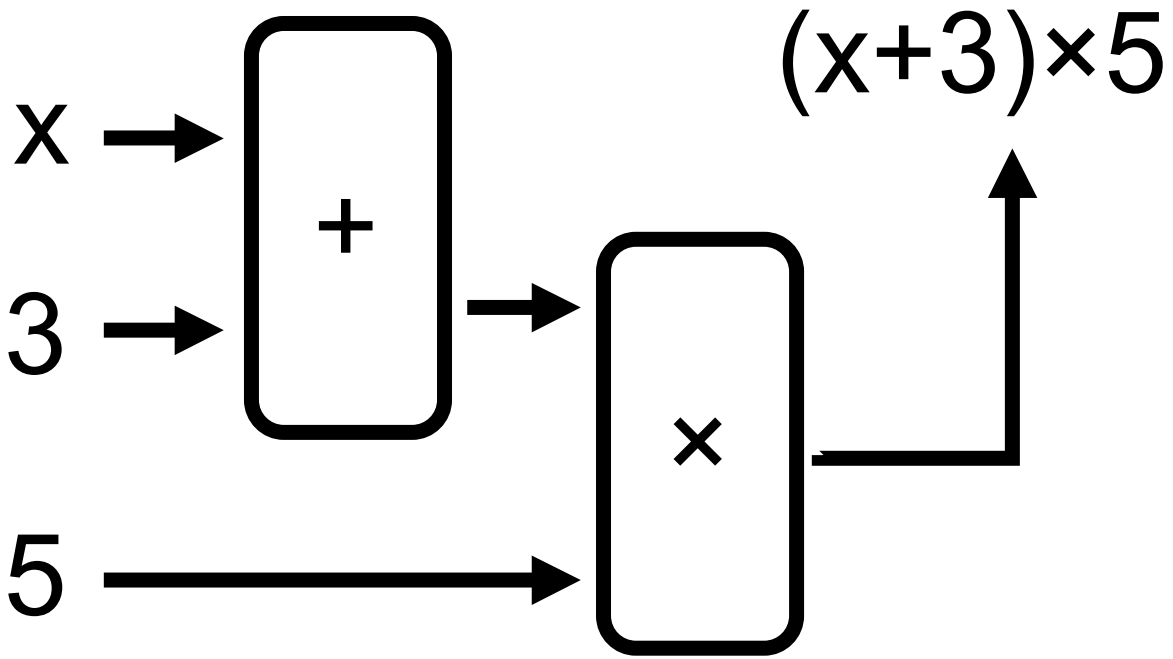
- 「穴」のあいたプロセスとその結果，両方．



変数を含む数式

$$(x + 3) \times 5$$

- 「穴」のあいたプロセスとその結果，両方．



プロセスとその結果

- 変数を含む数式は，穴のあいたプロセスと，その結果の両方を表わすので紛らわしい．
- 型付ラムダ計算では，それらを区別する．

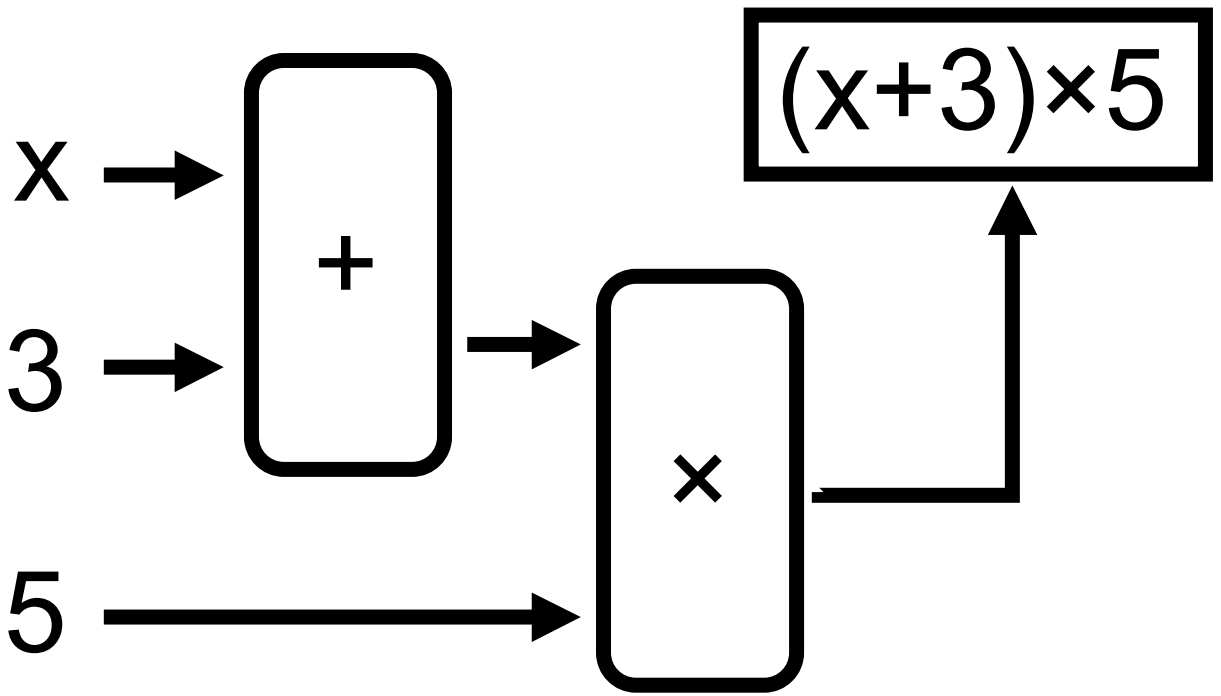
$$(x + 3) \times 5$$

- これ自体は，穴のあいたプロセスの結果を表わすこととする．
- だから，何らかの「不特定な数」．

変数を含む式

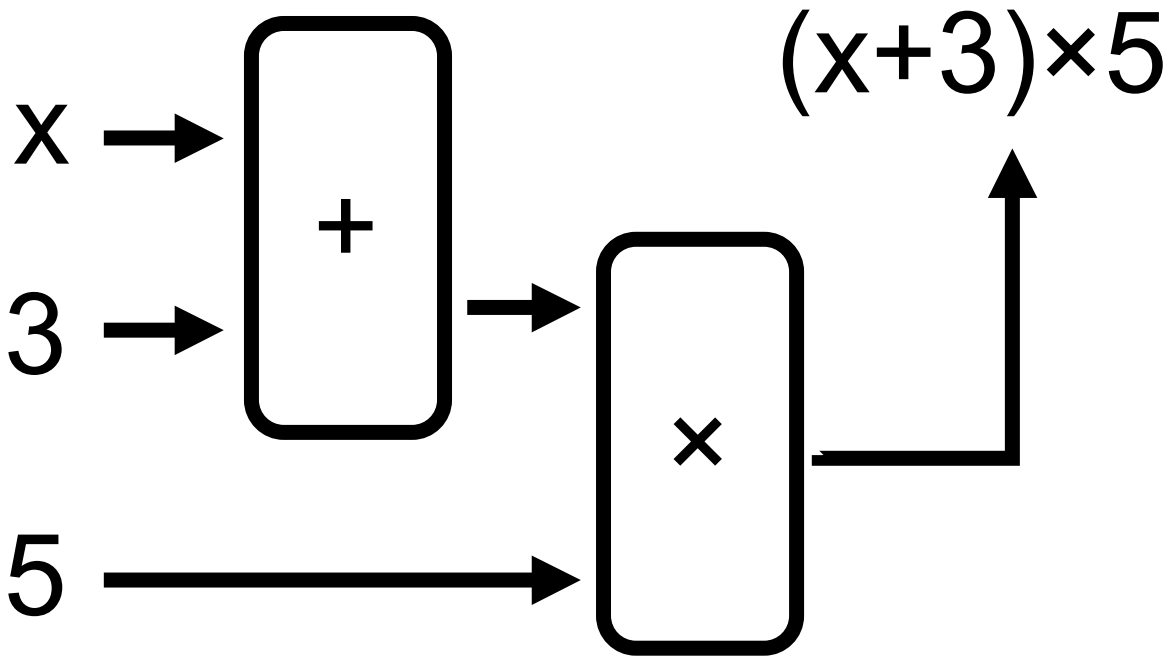
$$(x + 3) \times 5$$

- 結果の方を表わす．



プロセス

- プロセスそのものはどう表わす？



ラムダ記法

- プロセスそのものはどう表わすか.

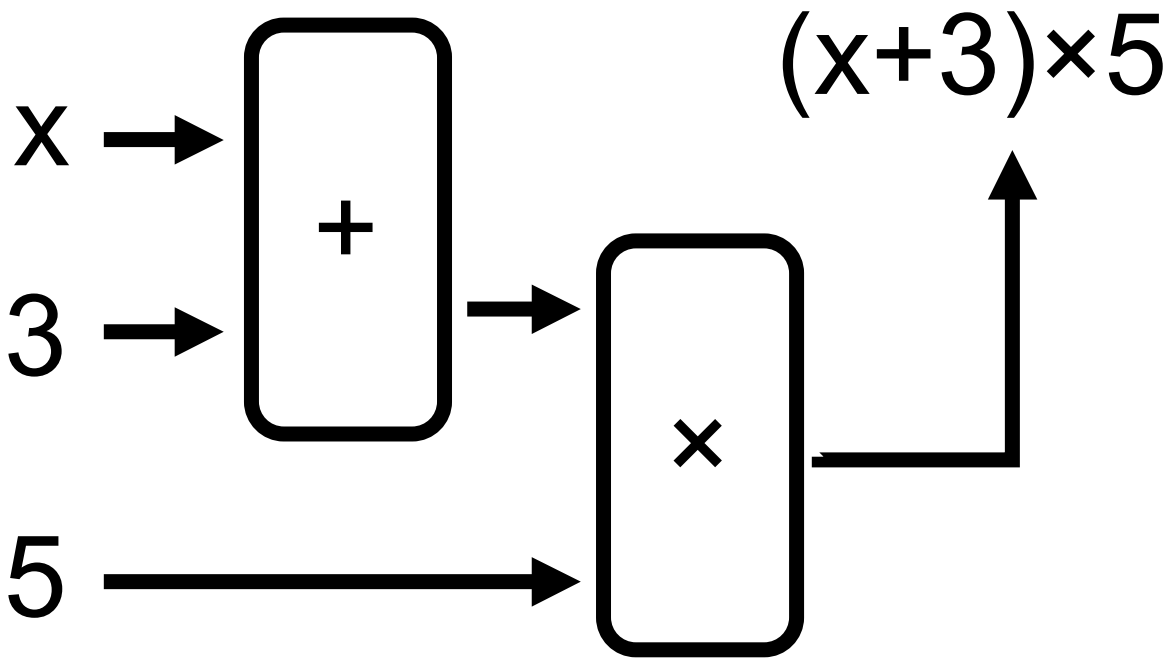
$$\lambda x.((x + 3) \times 5)$$

  と表わす.「ラムダ記法」という.
- 関数を表わすプログラム.

ラムダ記法

$$\lambda x.((x + 3) \times 5)$$

- 穴のあいたプロセスそのものを表わす．

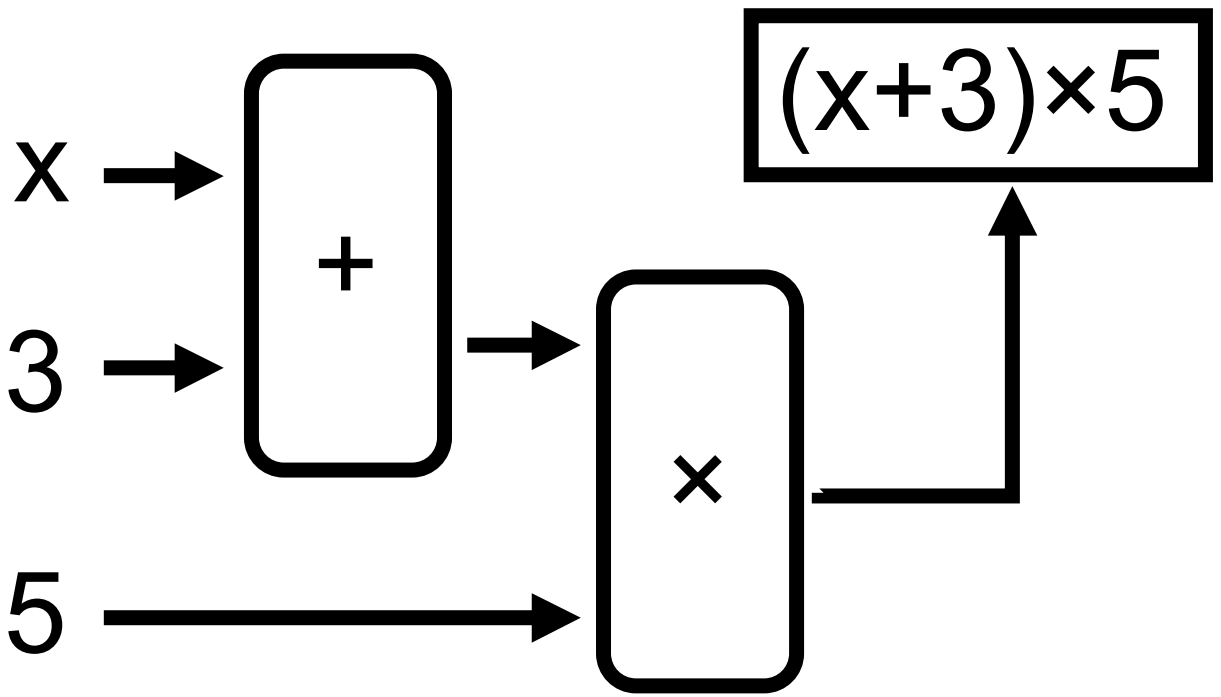


ラムダ記法

$$\lambda x.((x + 3) \times 5)$$

- 入力が x，出力が $(x + 3) \times 5$ となる関数．
- 入力と出力と関数そのもの．違う種類のデータ．
- それらの「型」について考えていく．

プロセスの出力に注目



不特定の数とは？

$$(x + 3) \times 5$$

- これは，何らかの不特定な数を表わす．
- 不特定とはどういうことか？—— x の値が決まらないかぎり，全体の値も決まらない．
- でもどういう「種類」の値になるかは，考えることができる．

不特定の数とは？

$$(x + 3) \times 5$$

- x にはどんな種類のものが入るか？—— 人間や動物なんかのモノではない．数だ．
- ここでは，整数だけを入れることにしよう．つまり，x は整数だとする．

不特定の数とは？

$$(x + 3) \times 5$$

- x が整数だとしたら，$(x + 3) \times 5$ は？
- もちろん整数．
- $(x + 3) \times 5$ は，値ははっきりしないけれども，x が整数だという仮定のもとでは，整数になる．

データの持つ型

- x は整数という種類に属する．これを

$$x : \mathbf{int}$$

と書く．これを「x は整数型をもつ」,「x は整数という型を持つ」と読む．

※ **int は，整数** integer **の** int**．**

データの持つ型

$$(x + 3) \times 5$$

- では，これはどういう種類のものか？
- いま x は整数なので，これも整数だ．つまり，

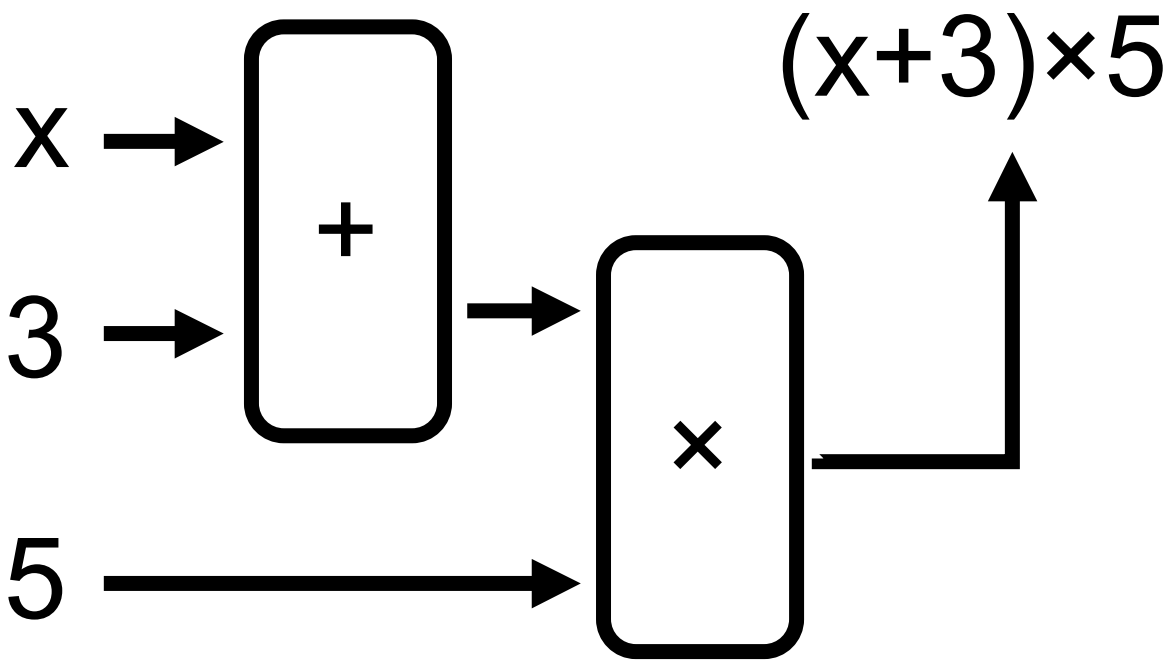
$$(x + 3) \times 5 : \mathrm{int}$$

- 不特定だが，$x : \mathrm{int}$ とすれば，これも整数．

プロセスとしての関数

$$\lambda x.((x + 3) \times 5)$$

- これはプロセスそのものを表わす．



プロセスとしての関数

$$\lambda x.((x + 3) \times 5)$$

- いま，入力部分の穴 x は整数，そして結果 $(x + 3) \times 5$ も整数．
- だから，このプロセスは，整数が入力として与えられたら，整数を出力として返すプロセス，すなわち関数．こう表わす：

$$\lambda x.((x + 3) \times 5) : \mathbf{int} \to \mathbf{int}$$

プロセスとしての関数

$$\lambda x.((x + 3) \times 5) : \mathbf{int} \to \mathbf{int}$$

- これが関数の「型」.
- 整数を入力とし，整数を出力とする関数．関数の型は，入力と出力の型で決まる.
- 一般に，型 A のデータを入力とし，型 B のデータを出力とする関数の型は，

$$A \to B$$

で表わす.

関数を作るとは

- 入力 x の型を決めると，出力 $(x + 3) \times 5$ の型が決まる．すると関数の型が決まる．

$$x : \mathrm{int} \Rightarrow (x + 3) \times 5 : \mathrm{int}$$
$$\Downarrow$$
$$\lambda x.((x + 3) \times 5) : \mathrm{int} \to \mathrm{int}$$

- これが関数を表わすプログラムの作り方．

ラムダ抽象

$$\dfrac{\dfrac{\begin{array}{c}[x : \mathbf{int}] \\ \vdots \\ (x + 3) \times 5 : \mathbf{int}\end{array}}{\lambda x.((x + 3) \times 5) : \mathbf{int} \to \mathbf{int}}}{}$$

- $\lambda x.((x + 3) \times 5)$ というプログラムを作る過程は，このように書くことができる．
- プログラムの「構成図」．

抽象規則

$$\frac{\begin{array}{c}[x : A] \\ \mathcal{D} \\ M : B\end{array}}{\lambda x.M : A \to B}\ (\text{abs})$$

- x が型 A をもつという仮定のもとで，
- 型 B をもつ M が構成できたなら，
- $\lambda x.M$ というプログラムが構成できる．

抽象規則

$$\frac{\begin{array}{c}[x : A] \\ \mathcal{D} \\ M : B\end{array}}{\lambda x.M : A \to B}\ (\text{abs})$$

- $\lambda x.M$ の型は $A \to B$，
  すなわち A から B への関数である．
- このとき，$x : A$ という仮定は消去される．

ラムダ抽象

$$\frac{\begin{array}{c}[x : A] \\ \mathcal{D} \\ M : B\end{array}}{\lambda x.M : A \to B}\ (\text{abs})$$

- 関数（プロセス）を抽象（抽出）するので，

これを「ラムダ抽象」という

(λ-abstraction)

今度は入力

- 関数を作ったら，入力したくなるのが人情．
- ただし，関数を作る際にはその「型」，入力と出力のデータの種類をちゃんと決めた．

⇒ 型の不整合が起こらないような入力の仕方でなければいけない．

入力の表わし方

$$\lambda x.((x + 3) \times 5) : \mathbf{int} \to \mathbf{int}$$

- 2 を入力してみよう．
- 入力は，後ろにくっつけて書く：

$$(\lambda x.((x + 3) \times 5))\ 2$$

- 関数 $\lambda x.((x + 3) \times 5)$ に 2 を入力する，という段取りを表わすプログラム．

型の不整合

$$2\ (\lambda x.((x + 3) \times 5))$$

- 2 に $\lambda x.((x + 3) \times 5)$ を入力？おかしい．

$$(\lambda x.((x + 3) \times 5))\ \times$$

- $\lambda x.((x + 3) \times 5)$ に × を入力？何かおかしい．

不適切な入力が行われている
型の不整合が起こっている

何が入力できるか

$$2\ (\lambda x.((x + 3) \times 5))$$

- M に N を入力できるためには，M は関数でなければならない．すなわち，M の型は

$$M : A \to B$$

  という形でなければならない．
- でも，$2 : \mathrm{int}$ のはず．何も入力できない．だから，不適切な入力．型が不整合．

何が入力できるか

$$(\lambda x.((x + 3) \times 5)) \times$$

- $\lambda x.((x + 3) \times 5)$ は確かに関数．でもその型は，

$$\lambda x.((x + 3) \times 5) : \mathrm{int} \to \mathrm{int}$$

- 入力できるのは整数だけ．$\times$ は整数ではない．
  だから，不適切な入力．型が不整合．

入力のための「適用規則」

$$\frac{M : A \to B \quad N : A}{MN : B}\ (\text{app})$$

- M が $A \to B$ という型を持つなら，
  それに A という型のデータを入力できる．
  できるプログラムの型は B である．

入力のための「適用規則」

$$\frac{M : A \to B \quad N : A}{MN : B}\ (\text{app})$$

- それ以外の入力は不可．不整合．
- 「適用規則」と呼ばれる．

- **われわれの問題：**

  型の不整合が，絶対に起こらないようなプログラムの書き方とは？

型

- 型：データの種類．
- int のような，入力になる（入力にしかならない）データ．「基本型」と呼ばれる．
  - 実際のプログラミングでは，他に文字 (char) 型やブーリアン (Boolean) 型など．
- 関数型：入力と出力の型から決まる．
  - $\mathrm{int} \to \mathrm{int}$，$\mathrm{int} \to (\mathrm{int} \to \mathrm{int})$，$(\mathrm{int} \to \mathrm{int}) \to (\mathrm{int} \to \mathrm{int})$ とか，どんどんできる．

基本的な関数

- ちなみに，基本的な関数，+ や × の型は，

$$+ : \mathrm{int} \to (\mathrm{int} \to \mathrm{int})$$
$$\times : \mathrm{int} \to (\mathrm{int} \to \mathrm{int})$$

- 整数を二回入力すると，整数を出力するということ．

プログラムの構成規則

$$\frac{\begin{array}{c}[x : A] \\ \mathcal{D} \\ M : B\end{array}}{\lambda x.M : A \to B} \qquad \frac{M : A \to B \quad N : A}{MN : B}$$

- 基本的な型付データ（関数と整数など）から出発して，型を明示した構成規則に従ってプログラムを作っていく

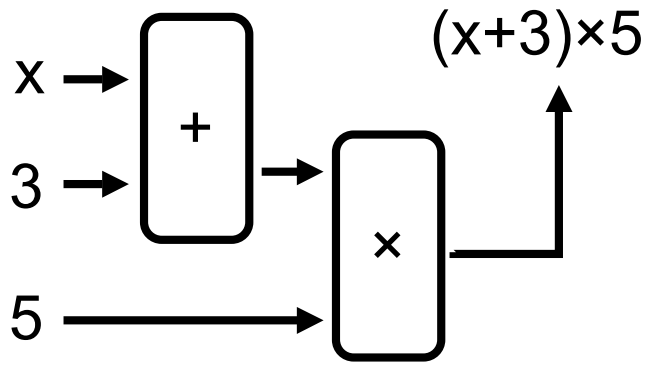
$\Rightarrow$ 型の不整合は起こらないはず．

例：$\lambda x.((x + 3) \times 5)$ の構成図

$$\dfrac{\dfrac{\dfrac{\dfrac{\_ + \_ : \mathrm{i} \to (\mathrm{i} \to \mathrm{i}) \quad [x : \mathrm{i}]}{x + \_ : \mathrm{i} \to \mathrm{i}} \quad 3 : \mathrm{i}}{(x + 3) : \mathrm{i}} \quad \_ \times \_ : \mathrm{i} \to (\mathrm{i} \to \mathrm{i})}{(x + 3) \times \_ : \mathrm{i} \to \mathrm{i}} \quad 5 : \mathrm{i}}{((x + 3) \times 5) : \mathrm{i}}$$

$$\lambda x.((x + 3) \times 5) : \mathrm{i} \to \mathrm{i}$$

- 「int」を「i」に省略．
- 大まかに下の図と対応する．



構成図

- われわれが絵で書くような計算プロセス＝関数の入力と出力の組み合わせ．
- 型の不整合が起こらないように，型を明示した構成図で組み合わせ，プロセスを形成する．
- 構成図の最後の行が作られたプログラム．そのプログラムの型が明示されている．

型付ラムダ計算

- $\lambda x.((x + 3) \times 5)$ のようなラムダ記法
- 型を明示した構成規則に従って構成図を書くことで，プログラムを作る．
- このようなプログラムの書き方を

型付ラムダ計算
(typed lambda calculus)

という．

- とはいえ，この呼び名はまだ不正確．
- なぜなら，まだプログラムを書いただけだから．プログラムは「実行されてナンボ」．
- 「型付ラムダ計算」は，プログラムの実行の仕方まで含んだ呼び名．
- というわけで，書いたプログラムがどのように実行されるか，見ていこう．

今までやったこと

- 関数を作って，それに何かを入力するプログラムを書いた．例えば，

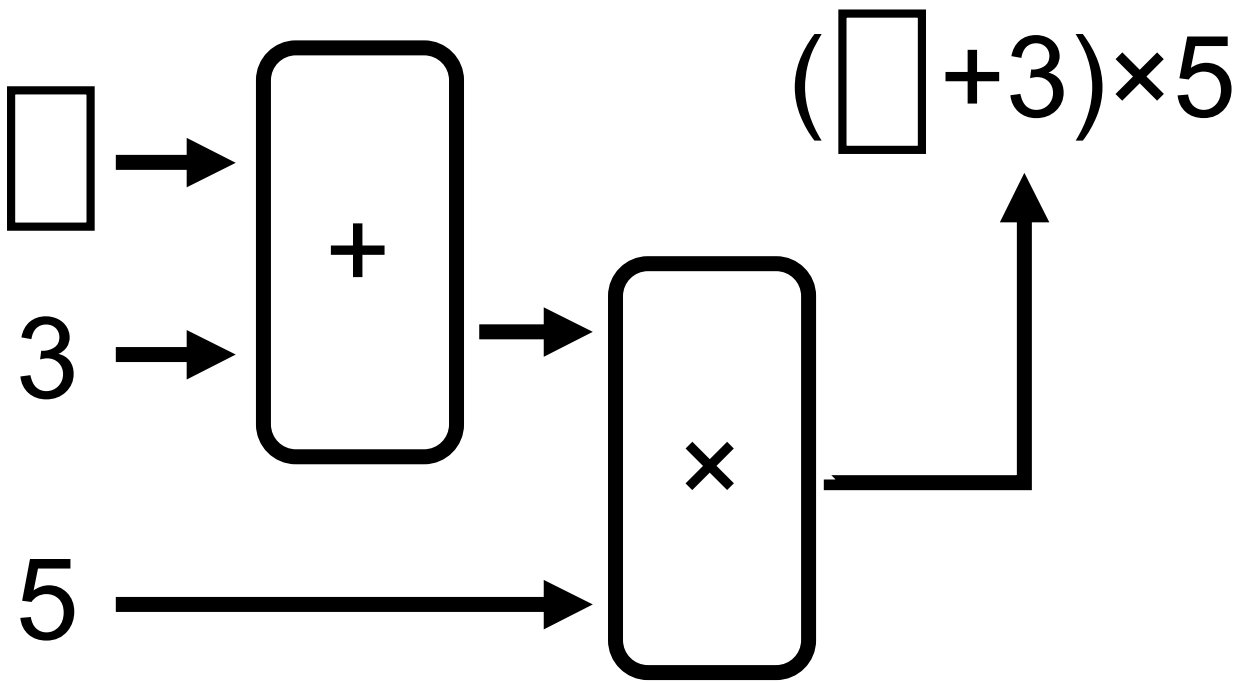
$$(\lambda x.((x + 3) \times 5))\ 2$$

- 入力したら，実行してみたくなるのが人情．

関数

$$\lambda x.((x + 3) \times 5)$$

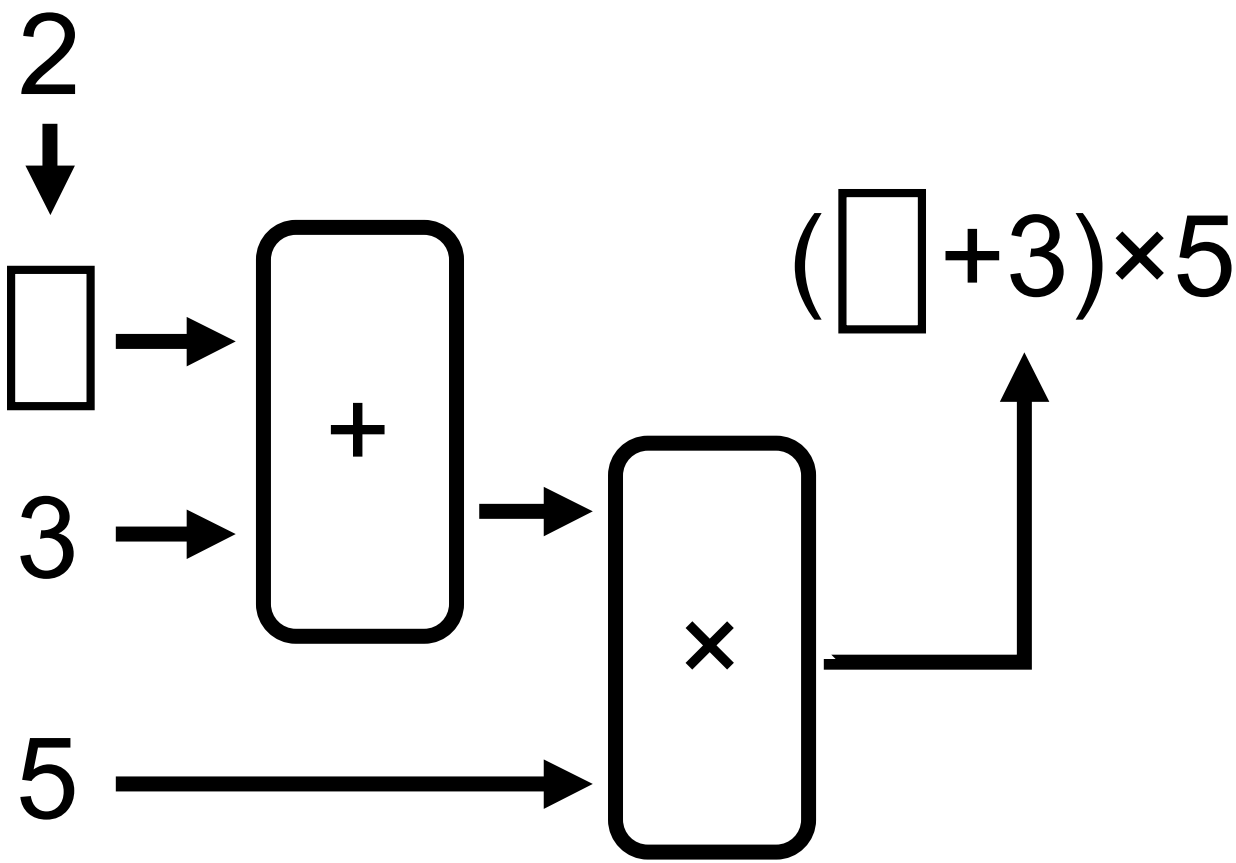
- これはプロセスそのものを表わす．



関数への入力

$$(\lambda x.((x + 3) \times 5))\ 2$$

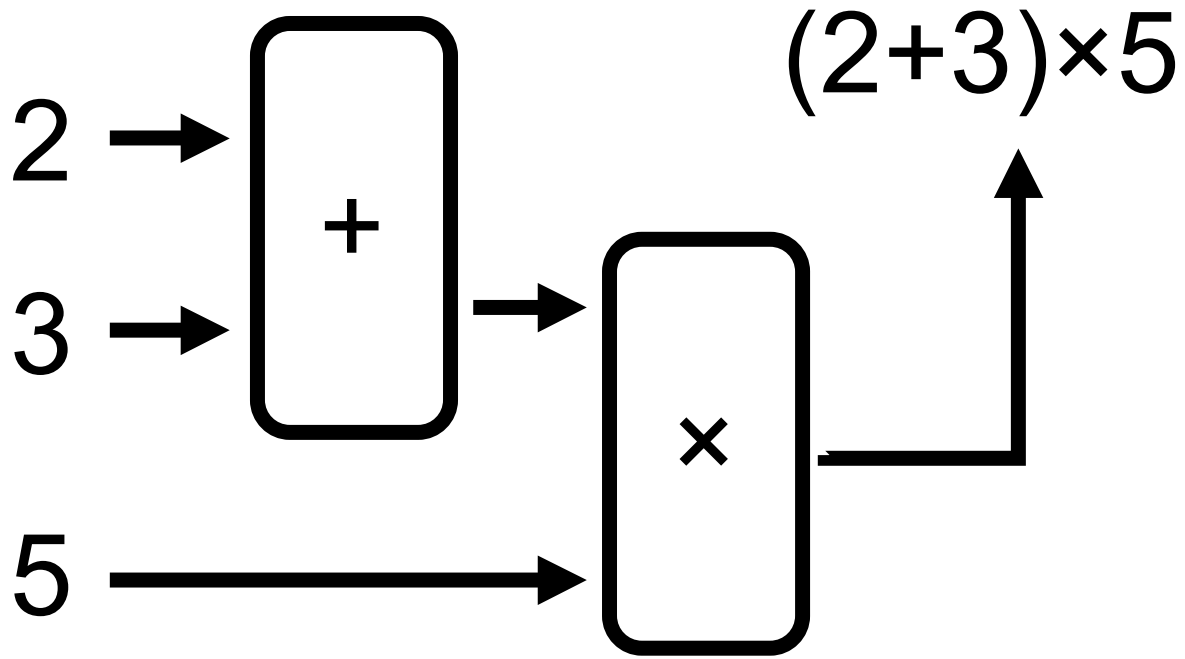
- それに入力しろってことだから…



実行すると

$$(\lambda x.((x + 3) \times 5))\ 2 \longmapsto (2 + 3) \times 5$$

- こうなるはず．



実行は代入

- 関数は穴のあいたプロセス．
- 実行＝穴のところに入力を入れろ．
- 実行＝xのところに「代入」しろ．

$$
\begin{aligned}
(\lambda x.((x + 3) \times 5))\ 2 \\
\longmapsto (2 + 3) \times 5 \\
= 25
\end{aligned}
$$

いろんな入力で実行してみる

$$(\lambda x.((x + 3) \times 5))\ 2$$
$$\longmapsto (2 + 3) \times 5 = 25$$

$$(\lambda x.((x + 3) \times 5))\ 3$$
$$\longmapsto (3 + 3) \times 5 = 30$$

$$(\lambda x.((x + 3) \times 5))\ 4$$
$$\longmapsto (4 + 3) \times 5 = 35$$
$$\vdots$$

β 簡約

- **関数に入力して実行すると，代入が起こる．実行とは代入のことだ．**

$$(\lambda x.M)N \longmapsto M[x := N]$$

これを「ベータ簡約」という．

(β-reduction)

プログラムを書いて，実行する

$$\frac{\begin{array}{c}[x : A]\\ \mathcal{D}\\ M : B\end{array}}{\lambda x.M : A \to B} \qquad \frac{M : A \to B \quad N : A}{MN : B}$$

$$(\lambda x.M)N \longmapsto M[x := N]$$

- 型の不整合を防ぐための構成規則．
- 実行＝代入＝ベータ簡約

実行しても大丈夫か？

$$\frac{\begin{array}{c}[x : A] \\ \mathcal{D} \\ M : B\end{array}}{\lambda x.M : A \to B} \qquad \frac{M : A \to B \quad N : A}{MN : B}$$

$$(\lambda x.M)N \longmapsto M[x := N]$$

- でもベータ簡約では，型の話が出ていない．
- 実行したときに，型の不整合は起こらないのだろうか？

どういう問題か

$$\dfrac{\dfrac{\vdots}{\lambda x.M : B \to C} \quad \dfrac{\dfrac{\vdots}{\lambda y.N_1 : A \to B} \quad \dfrac{\vdots}{N_2 : A}}{(\lambda y.N_1)N_2 : B}}{(\lambda x.M)((\lambda y.N_1)\ N_2) : C}$$

- こういうプログラムを構成規則に従って書いたとする．
- $\lambda y.N_1$ に N_2 を入力し，その結果を $\lambda x.M$ に入力せよ，というプログラム．

どういう問題か

$$(\lambda x.M)((\lambda y.N_1)\ N_2)$$

- $\lambda y.N_1$ に N_2 を入力し，その結果を $\lambda x.M$ に入力せよ，というプログラム．
- これを 1 ステップ実行してみよう．

$$(\lambda x.M)((\lambda y.N_1)N_2)$$
$$\mapsto (\lambda x.M)(N_1[y := N_2])$$

- 次は，$\lambda x.M$ への $N_1[y := N_2]$ の入力．

どういう問題か

$$\frac{\begin{array}{cc}\vdots & \vdots \\ \lambda x.M : B \to C & N_1[y := N_2] : B\,?\end{array}}{(\lambda x.M)(N_1[y := N_2]) : C\,?}$$

- 実行してできた $N_1[y := N_2]$ の型は B だろうか？
- もし，実行して型が変わってしまったら，$\lambda x.M$ への入力はできなくなる．つまり，型の不整合になってしまう．

問題

- 「実行プロセスの途中で，型の不整合は起こらないだろうか」という問題は，次の問題に帰着できる．

$$(\lambda x.M)N : B \text{ なら}$$
$$M[x := N] : B \text{ だろうか？}$$

- つまり，

ベータ簡約は型を変えないだろうか？

答え：変えない

$$(\lambda x.M)N \longmapsto M[x := N]$$

- $(\lambda x.M)N$ は適用の形．構成図は次のような形をしているはず．

$$\dfrac{\begin{array}{cc}\vdots & \mathcal{D}_1 \\ \lambda x.M : A \to B & N : A\end{array}}{(\lambda x.M)N : B}$$

答え：変えない

$$(\lambda x.M)N \longmapsto M[x := N]$$

- 左側はラムダ抽象の形．だから構成図は次のような形のはず．

$$\dfrac{\dfrac{\begin{array}{c}[x : A]\\ \mathcal{D}_0\\ M : B\end{array}}{\lambda x.M : A \to B} \quad \begin{array}{c}\mathcal{D}_1\\ N : A\end{array}}{(\lambda x.M)N : B}$$

答え：変えない

$$(\lambda x.M)N \longmapsto M[x := N]$$

- 実行＝ベータ簡約は代入．
- 代入を，プログラム構成図で考えてみよう．

$$\dfrac{\dfrac{\begin{array}{c}[x : A]\\ \mathcal{D}_0\\ M : B\end{array}}{\lambda x.M : A \to B} \quad \begin{array}{c}\mathcal{D}_1\\ N : A\end{array}}{(\lambda x.M)N : B}$$

答え：変えない

$$(\lambda x.M)N \longmapsto M[x := N]$$

- 実行＝ベータ簡約は代入．
- 代入を，プログラム構成図で考えてみよう．

$$\mathcal{D}_1$$
$$N : A$$
$$\mathcal{D}_0[x := N]$$
$$M[x := N] : B$$

ベータ簡約＝構成図の代入

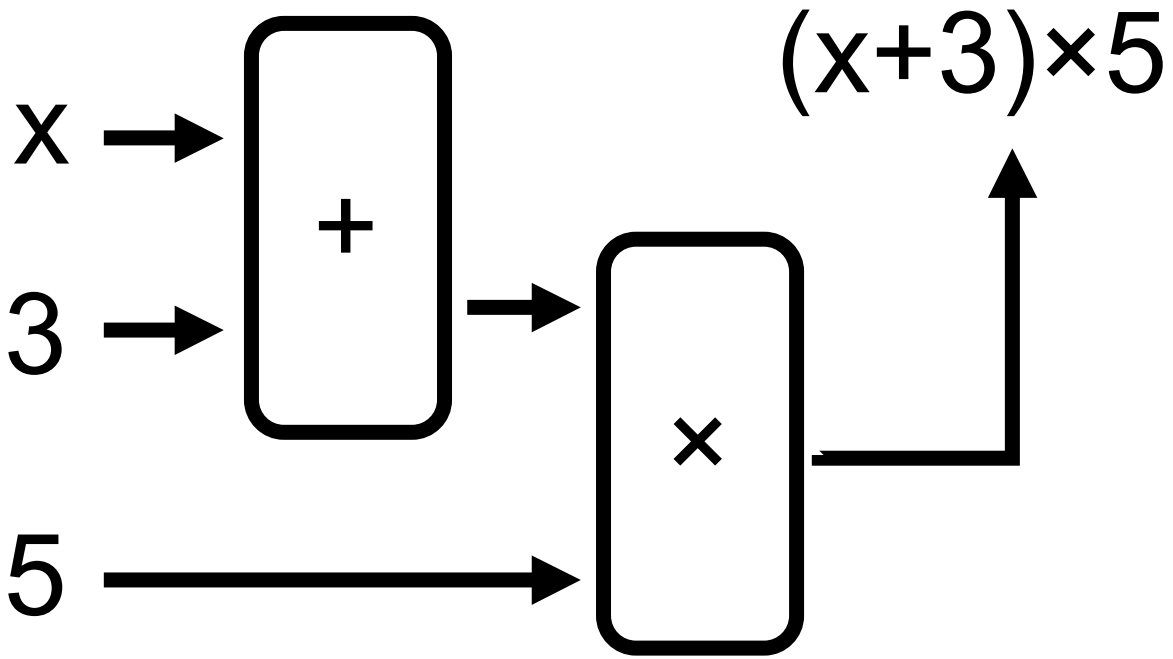
$$\dfrac{\dfrac{\begin{array}{c}[x:A]\\ \mathcal{D}_0\\ M:B\end{array}}{\lambda x.M:A\to B}\quad \begin{array}{c}\mathcal{D}_1\\ N:A\end{array}}{(\lambda x.M)N:B} \longmapsto \begin{array}{c}\mathcal{D}_1\\ N:A\\ \mathcal{D}_0[x:=N]\\ M[x:=N]:B\end{array}$$

- ベータ簡約は，元の構成図の x を N で置き換える，という作業と考えられる．
- x と N はこのとき絶対に同じ型．だから置き換えても，全体の型は変わらない．

- ベータ簡約は型を変えないことがわかった．
- それゆえ，構成規則だけに従ってプログラムを書けば，実行の途中で型の不整合は起こらないことが保証された．

まとめ：型付ラムダ計算

- 数式が表わすような計算プロセス．入力と出力の組み合わせ．



まとめ：型付ラムダ計算

$$\frac{\begin{array}{c}[x : A] \\ \mathcal{D} \\ M : B\end{array}}{\lambda x.M : A \to B} \qquad \frac{M : A \to B \quad N : A}{MN : B}$$

- 適切な入力・出力の組み合わせと，それを用いた関数の抽出．

まとめ：型付ラムダ計算

$$\dfrac{\dfrac{\dfrac{\dfrac{\_ + \_ : \mathrm{i} \to (\mathrm{i} \to \mathrm{i}) \quad [x : \mathrm{i}]}{x + \_ : \mathrm{i} \to \mathrm{i}} \quad 3 : \mathrm{i}}{(x + 3) : \mathrm{i}}}{\dfrac{\_ \times \_ : \mathrm{i} \to (\mathrm{i} \to \mathrm{i}) \qquad (x + 3) : \mathrm{i}}{(x + 3) \times \_ : \mathrm{i} \to \mathrm{i}} \quad 5 : \mathrm{i}}}{((x + 3) \times 5) : \mathrm{i}}}{\lambda x.((x + 3) \times 5) : \mathrm{i} \to \mathrm{i}}$$

- 絵で何となく書いていた計算プロセスを，厳密な構成図で表わすことができる．

まとめ：型付ラムダ計算

$$(\lambda x.M)N \longmapsto M[x := N]$$

$$\dfrac{\dfrac{\begin{array}{c}[x : A]\\ \mathcal{D}_0\\ M : B\end{array}}{\lambda x.M : A \to B} \quad \begin{array}{c}\mathcal{D}_1\\ N : A\end{array}}{(\lambda x.M)N : B} \longmapsto \begin{array}{c}\mathcal{D}_1\\ N : A\\ \mathcal{D}_0[x := N]\\ M[x := N] : B\end{array}$$

- 実行しても，型は変わらない．実行の途中で，型の不整合は絶対に起こらない．
- 厳密に書いた構成図を使って，それを確かめることができる．

まとめ：型付ラムダ計算

- 数学における，

  数式を用いた計算プロセスの設計
  関数の抽出と，関数への入力・計算

  の「形式化」．パターンを取り出す．

$$\frac{\begin{array}{c}[x : A] \\ \mathcal{D} \\ M : B\end{array}}{\lambda x.M : A \to B} \qquad \frac{M : A \to B \quad N : A}{MN : B}$$

$$(\lambda x.M)N \longmapsto M[x := N]$$

まとめ：型付ラムダ計算

- 数学では，これらの作業はほとんど意識されることなく，スムーズに行われている．
- それをあえて，意識的に取り出す．なぜスムーズに行われているかを考える．

⇒ 計算を機械（コンピュータ）に行わせることができる．人間の洞察なしに．

⇒ しかも，機械がスムーズに，つまりエラーなしに計算できる．そのことが厳密に確かめられる．

Recall

カリー・ハワード同型対応って何？

- 簡単に言うと

  数学における証明の構造と，
  コンピュータ・プログラムの構造が，
  ぴったり一致する

  ということ．

同型対応

$$\frac{\begin{array}{c}[x : A]\\ \mathcal{D}\\ M : B\end{array}}{\lambda x.M : A \to B} \qquad \frac{M : A \to B \quad N : A}{MN : B}$$

$$\Downarrow \Uparrow$$

$$\frac{\begin{array}{c}[A]\\ \mathcal{D}\\ B\end{array}}{A \to B} \qquad \frac{A \to B \quad A}{B}$$

同型対応

- プログラムの構成図のプログラムの部分をはぎとると，自然演繹の証明になる．
- 型＝命題（論理式）
- 抽象規則＝ならば導入則
- 適用規則＝ならば除去則

プログラムから証明

- 型付のプログラムが与えられれば，そこから構成図が復元でき，そこから自然演繹の証明が得られる．

$$\dfrac{\begin{array}{c}[x:A]\\ \mathcal{D}\\ M:B\end{array}}{\lambda x.M:A\to B} \quad\Leftrightarrow\quad \dfrac{\begin{array}{c}[A]\\ \mathcal{D}\\ B\end{array}}{A\to B}$$

- プログラムがラムダ抽象「$\lambda x.M$」の形なら，対応する証明は最後のステップで，ならば導入則を使っていることがわかる．

プログラムから証明

- 型付のプログラムが与えられれば，そこから構成図が復元でき，そこから自然演繹の証明が得られる．

$$\frac{M : A \to B \quad N : A}{MN : B} \Leftrightarrow \frac{A \to B \quad A}{B}$$

- プログラムが適用「$M\ N$」の形なら，対応する証明は最後のステップで，ならば除去則を使っていることがわかる．

同型対応

- 逆に，自然演繹の証明が与えられれば，プログラムが作れる．
- 証明の仮定 A を，$x : A$ に直して…
  あとは省略．
- つまり，証明＝プログラム．

例：証明からプログラム

$$\dfrac{\dfrac{\dfrac{\dfrac{[A \to (B \to C)] \quad [A]}{B \to C} \quad \dfrac{[A \to B] \quad [A]}{B}}{C}}{A \to C}}{(A \to B) \to (A \to C)}}{(A \to (B \to C)) \to ((A \to B) \to (A \to C))}$$

- これを…

例：証明からプログラム

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[x : A \to (B \to C)] \quad [y : A]}{xy : B \to C} \quad \dfrac{[z : A \to B] \quad [y : A]}{zy : B}}{(xy)(zy) : C}}{\lambda y.((xy)(zy)) : A \to C}}{\lambda z \lambda y.((xy)(zy)) : (A \to B) \to (A \to C)}}{\lambda x \lambda z \lambda y.((xy)(zy)) : (A \to (B \to C)) \to ((A \to B) \to (A \to C))}$$

- こんな風に．

同型対応

型付ラムダ計算		自然演繹
型	=	命題
抽象規則	=	ならば導入則
適用規則	=	ならば除去則
プログラム	=	証明

- めでたしめでたし？

ベータ簡約はどこへいった？

$$(\lambda x.M)N \longmapsto M[x := N]$$

$$\dfrac{\dfrac{\begin{array}{c}[x : A]\\ \mathcal{D}_0\\ M : B\end{array}}{\lambda x.M : A \to B} \quad \begin{array}{c}\mathcal{D}_1\\ N : A\end{array}}{(\lambda x.M)N : B} \longmapsto \begin{array}{c}\mathcal{D}_1\\ N : A\\ \mathcal{D}_0[x := N]\\ M[x := N] : B\end{array}$$

- プログラムの「実行」という要素．
- 「ベータ簡約＝構成図の代入」という操作．
- これに対応する操作なんてあるの？

ベータ簡約はどこへいった？

$$\dfrac{\dfrac{\begin{array}{c}[x:A]\\ \mathcal{D}_0\\ M:B\end{array}}{\lambda x.M:A\to B}\quad \begin{array}{c}\mathcal{D}_1\\ N:A\end{array}}{(\lambda x.M)N:B} \longmapsto \begin{array}{c}\mathcal{D}_1\\ N:A\\ \mathcal{D}_0[x:=N]\\ M[x:=N]:B\end{array}$$

- プログラムの部分をはぎとってみよう．

ベータ簡約はどこへいった？

$$\dfrac{\dfrac{\begin{array}{c}[A]\\ \mathcal{D}_0\\ B\end{array}}{A \to B} \quad \begin{array}{c}\mathcal{D}_1\\ A\end{array}}{B} \quad \longmapsto \quad \begin{array}{c}\mathcal{D}_1\\ A\\ \mathcal{D}_0\\ B\end{array}$$

- 何かできたけど，これって意味あるの？
- あります！

証明の回り道

$$\dfrac{\dfrac{\begin{array}{c}[A]\\ \mathcal{D}_0\\ B\end{array}}{A \to B} \quad \begin{array}{c}\mathcal{D}_1\\ A\end{array}}{B}$$

- この証明は，どういうものかを考えてみよう．
- ならば導入則の直後に，ならば除去則が使われている．これは証明の「回り道」である．

何をしているのか

$$\dfrac{\dfrac{[A]}{\mathcal{D}_0}}{\dfrac{B}{A \to B}} \quad \dfrac{\mathcal{D}_1}{A} \qquad \text{の} \qquad \dfrac{\mathcal{D}_1}{A}$$

- A が証明できる．A は正しい．

何をしているのか

$$\dfrac{\dfrac{\begin{array}{c}[A]\\ \mathcal{D}_0\\ B\end{array}}{A \to B} \quad \begin{array}{c}\mathcal{D}_1\\ A\end{array}}{B} \quad \text{の} \quad \dfrac{\begin{array}{c}[A]\\ \mathcal{D}_0\\ B\end{array}}{A \to B}$$

- A から B が導けるので，$A \to B$ だ．

何をしているのか

$$\dfrac{\dfrac{\begin{array}{c}[A]\\ \mathcal{D}_0\\ B\end{array}}{A \to B} \quad \begin{array}{c}\mathcal{D}_1\\ A\end{array}}{B}$$

- $A \to B$ と A が成り立っているので，B だ．

回り道

1. A が成り立っている．
2. A から B が導けるので，$A \to B$ だ．
3. $A \to B$ と A が成り立っているので，B だ．

- 回りくどくないか？欲しい結論は B．

1. A が成り立っている．
2. A から B が導ける．ゆえに B だ，で終わりじゃない？

証明の簡約

$$\dfrac{\dfrac{\begin{array}{c}[A]\\ \mathcal{D}_0\\ B\end{array}}{A \to B} \quad \begin{array}{c}\mathcal{D}_1\\ A\end{array}}{B} \qquad \longmapsto \qquad \begin{array}{c}\mathcal{D}_1\\ A\\ \mathcal{D}_0\\ B\end{array}$$

- 右側が意味するのは，「A が証明でき，さらにそこから B が導ける．おしまい」．
- シンプルな証明．回り道が消えた．

証明の簡約

$$\dfrac{\dfrac{\begin{array}{c}[A]\\ \mathcal{D}_0\\ B\end{array}}{A \to B} \quad \begin{array}{c}\mathcal{D}_1\\ A\end{array}}{B} \quad \longmapsto \quad \begin{array}{c}\mathcal{D}_1\\ A\\ \mathcal{D}_0\\ B\end{array}$$

- このような証明の回り道を消す操作：

証明の簡約
(reduction)

簡約が意味することは？

- 前回，こう言った：

命題の内容＝他の命題との帰結関係

- 簡約は，命題の内容の中でも「本質的な部分」を取り出すために役立つ．

簡約が意味することは？

$$\dfrac{\dfrac{\begin{array}{c}[A]\\ \mathcal{D}_0\\ B\end{array}}{A \to B} \quad \begin{array}{c}\mathcal{D}_1\\ A\end{array}}{B}$$

- 命題の内容を「何から帰結するか」という観点から考えていこう．
- ここでの B は何から帰結するか．その成立の根拠は何か．

簡約が意味することは？

$$\dfrac{\dfrac{\begin{array}{c}[A]\\ \mathcal{D}_0\\ B\end{array}}{A \to B} \quad \begin{array}{c}\mathcal{D}_1\\ A\end{array}}{B}$$

- B は $A \to B$ と A から帰結している．
  $A \to B$ と A が，B の成立の根拠．
- でも，この推論ステップは回り道だった．

簡約が意味することは？

$$\dfrac{\dfrac{\begin{array}{c}[A]\\ \mathcal{D}_0\\ B\end{array}}{A \to B} \quad \begin{array}{c}\mathcal{D}_1\\ A\end{array}}{B} \qquad \text{の} \qquad \begin{array}{c}A\\ \mathcal{D}_0\\ B\end{array}$$

- 本質的には B はすでにここで導かれている．
- B の成立のためには，A だけがあればよい．$A \to B$ はなくてもよい．

簡約が意味することは？

$$\dfrac{\dfrac{\begin{array}{c}[A]\\ \mathcal{D}_0\\ B\end{array}}{A \to B} \quad \begin{array}{c}\mathcal{D}_1\\ A\end{array}}{B} \quad \longmapsto \quad \begin{array}{c}\mathcal{D}_1\\ A\\ \mathcal{D}_0\\ B\end{array}$$

- B の成立のために，必要十分なものだけを取り出していく．余計なものをそぎ落とす．
- 簡約は，こういう意味を持つ．

同型対応の完成

$$\frac{\begin{array}{c}[x : A]\\ \mathcal{D}\\ M : B\end{array}}{\lambda x.M : A \to B} \qquad \frac{M : A \to B \quad N : A}{MN : B}$$

$$\Downarrow \Uparrow$$

$$\frac{\begin{array}{c}[A]\\ \mathcal{D}\\ B\end{array}}{A \to B} \qquad \frac{A \to B \quad A}{B}$$

同型対応の完成

$$\dfrac{\dfrac{\begin{array}{c}[x:A]\\ \mathcal{D}_0\\ M:B\end{array}}{\lambda x.M:A\to B}\quad \begin{array}{c}\mathcal{D}_1\\ N:A\end{array}}{(\lambda x.M)N:B} \longmapsto \begin{array}{c}\mathcal{D}_1\\ N:A\\ \mathcal{D}_0[x:=N]\\ M[x:=N]:B\end{array}$$

$$\Downarrow\Uparrow$$

$$\dfrac{\dfrac{\begin{array}{c}[A]\\ \mathcal{D}_0\\ B\end{array}}{A\to B}\quad \begin{array}{c}\mathcal{D}_1\\ A\end{array}}{B} \longmapsto \begin{array}{c}\mathcal{D}_1\\ A\\ \mathcal{D}_0\\ B\end{array}$$

同型対応

型付ラムダ計算		自然演繹
型	=	命題
抽象規則	=	ならば導入則
適用規則	=	ならば除去則
プログラム	=	証明
ベータ簡約	=	証明の簡約

論理と計算：形式化

- 同型対応：論理と計算，それぞれの形式化により，明らかになったこと．
- 自然演繹と型付ラムダ計算：それぞれに異なる問題意識，概念枠組．

- 自然演繹：証明にかんする考察を通じて，数学という研究，数学的命題の本性を探る．
- 型付ラムダ計算：コンピュータに，エラーの心配なしに計算をさせる．

論理と計算：形式化

- 論理と計算，両者の間にこれほどまでにきれいな対応が成り立つのはなぜか？

答え：よくわからない．

- いくつかの理由の考察については，照井 [2005] を参照．

論理と計算：形式化

- 現在の両者の概念枠組で，同型性を説明するのは難しいと思われる．

⇒ 両者を一挙に捉えることのできる，包括的な新しい概念枠組が必要（たぶん）．

論理と計算：形式化

- 形式化：実践の中に潜んでいる，当たり前の，それゆえにあまり意識されることのないパターンを，あえて明示的に取り出す．

⇒ 形式化するまでは思いもよらなかったことがわかる．従来の概念枠組に再考を迫る．

- 同型対応が教えてくれること：形式化は，新しい哲学的問題の発見につながる．

レポート課題

- カリー・ハワード同型対応はなぜ成り立つか．
- 証明のもつ，計算的・関数的性質についてまとめる．照井 [2005] を参照するとよい．
- 他の形式化．その目的と効果．
  - 例：科学が発見する自然法則（パターン）．
  - 例：店員の接客マニュアル．

おしまい！

おつかれさまでした
ありがとうございました