Curry-Howard の対応

亀山幸義

筑波大学コンピュータサイエンス専攻

2013 年

Curry-Howard の同型対応

直観主義の命題論理	(単純) 型付きラムダ計算
命題	型
論理記号 $\wedge, \vee, \supset$	型構成子 $\times, +, \rightarrow$
導出 (証明) $\wedge I, \wedge EL, \wedge ER$ $\vee IL, \vee IR, \vee E$ $\rightarrow I, \rightarrow E$	項と導出 (証明) $\mathtt{pair}, \mathtt{left}, \mathtt{right}$ $\mathtt{inl}, \mathtt{inr}, \mathtt{case}$ $\lambda, \mathtt{apply}$
回り道の除去 $\wedge$ の回り道除去 $\vee$ の回り道除去 $\supset$ の回り道除去	計算 $\mathtt{left}(\langle M, N\rangle) \rightarrow M$ etc. $\mathtt{case}(...) \rightarrow ...$ etc. $(\lambda x : A.M)N \rightarrow M\{x := N\}$

導出の対応 (例)

$$\dfrac{\dfrac{\dfrac{}{A \wedge B \vdash A \wedge B}\,assume}{A \wedge B \vdash B}\,\wedge ER \quad \dfrac{\dfrac{}{A \wedge B \vdash A \wedge B}\,assume}{A \wedge B \vdash A}\,\wedge EL}{\dfrac{A \wedge B \vdash B \wedge A}{\vdash (A \wedge B) \supset (B \wedge A)}\,\supset I}\,\wedge I$$

$$\dfrac{\dfrac{\dfrac{}{x : A \times B \vdash x : A \times B}\,var}{x : A \times B \vdash \mathtt{right}(x) : B}\,\mathtt{right} \quad \dfrac{\dfrac{}{x : A \times B \vdash x : A \times B}\,var}{x : A \times B \vdash \mathtt{left}(x) : A}\,\mathtt{left}}{\dfrac{x : A \times B \vdash N : B \times A}{\vdash M : (A \times B) \rightarrow (B \times A)}\,\lambda}\,\mathtt{pair}$$

ただし、$M = \lambda x : (A \times B).\ \langle \mathtt{right}(x), \mathtt{left}(x)\rangle$.
$N = \langle \mathtt{right}(x), \mathtt{left}(x)\rangle$.

命題＝型 (Propositions as Types)

- 問. $A \supset (B \supset A)$ は証明可能な命題か?
- 答. YES. なぜなら、$A \rightarrow (B \rightarrow A)$ という型を持つラムダ式があるから。
  $K = \lambda x.\lambda y.x$.
- 問. $(A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C))$ は証明可能な命題か?
- 答. YES. なぜなら、$(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$ という型を持つラムダ式があるから。
  $S = \lambda x.\lambda y.\lambda z.\ (xz)(yz)$.

対応についての微調整

これまでの授業で説明してこなかったこと： 本当に、対応しているのか？

- $\bot$ という命題 (偽をあらわす命題) に対応する型は？
- $\neg$ という論理記号について、対応する型構成子がない。

$\bot$ について

命題 $\bot$ に対応する型は？

- $\bot$ は常に偽をあらわす。よって，証明はない.
- Curry-Howard の対応により，$\bot$ に対応する型は，その型をもつ項が 1 つも存在しないものである.
- つまり,「空の型」が $\bot$ に対応する.

「空の型」はどうやって使えばよいのか？
Abort 命令の型: $1 + 2 * (\mathit{abort}\ 0) + 4 \rightsquigarrow \mathit{abort} 0$.
プログラムのどんなところからでも、abort によって一気にプログラムを終了できる。

$$\frac{\Gamma \vdash M : \bot}{\Gamma \vdash \mathrm{abort} M : A}$$

「起き得ないことが起きたら abort する」という意味.

$\neg$ について

直観主義論理では、$\neg A$ は $A \supset \bot$ のこと (別名、マクロ定義)。
対応する型システムでは、$A \to \mathtt{empty}$。(empty は空の型)

型 $A \to \mathtt{empty}$ の読み方: A 型の要素をもらって、(何かを計算するが、その途中で)abort する関数の型。

証明＝プログラム

- Curry-Howard の同型対応のもとで、証明 (導出) と項 (プログラム) は同じ。
- プログラムは、一種の証明。
- 証明は、プログラムと見なせる ???

証明からのプログラム合成 (抽出)

与えられた自然数が、偶数か奇数かを判定するプログラムの仕様:

$$\forall x \in \mathtt{Nat}.\exists y \in \mathtt{Nat}.(\mathit{Even}(x) \wedge y = 0) \vee (\mathit{Odd}(x) \wedge y = 1)$$

この仕様の証明 (by 数学的帰納法):

$$\dfrac{\dfrac{\vdots}{\vdash \mathit{Even}(0) \wedge 0 = 0}}{\dfrac{\vdash M(0,0)}{\vdash \exists y.\ M(0,y)}} \quad \dfrac{\dfrac{\dfrac{\vdots}{\mathit{Even}(x) \wedge z = 0 \vdash M(x+1,1)}}{\mathit{Even}(x) \wedge z = 0 \vdash \exists y.\ M(x+1,y)} \quad \dfrac{\dfrac{\vdots}{\mathit{Odd}(x) \wedge z = 1 \vdash M(x+1,0)}}{\mathit{Odd}(x) \wedge z = 1 \vdash \exists y.\ M(x+1,y)}}{\dfrac{M(x,z) \vdash \exists y.\ M(x+1,y)}{\exists y.M(x,y) \vdash \exists y.\ M(x+1,y)}}$$

$$\vdash \forall x.\exists y.\ M(x,y)$$

$M(x,y) = (\mathit{Even}(x) \wedge y = 0) \vee (\mathit{Odd}(x) \wedge y = 1)$ とする。

証明からのプログラム合成 (抽出)-続き

- 前ページの証明 (導出) は、プログラムである!

$$\vdots$$
$$\vdash \forall x.\exists y.\ M(x,y)$$

- どうやってプログラムとして使うか?
- $x = 5$ のときの、y の値を知りたいとき：

$$\dfrac{\dfrac{\vdots}{\vdash \forall x.\exists y.\ M(x,y)}}{\vdash \exists y.\ M(5,y)}\ \forall E$$

- この証明に対する「回り道の除去」を適用する。(ここでは省略するが、数学的帰納法に対する回り道の除去ルールも決まっている。)

証明からのプログラム合成 (抽出)-続き

- 回り道がない証明 (正規な証明) は、以下の形をしている。

$$\dfrac{\dfrac{\vdots}{\vdash M(5,N)}}{\vdash \exists y.\ M(5,y)}\ \exists I$$

この N という項が、このプログラムの出力結果である。(最初の証明が正しければ、$N = 1$ となっているはずである。)

証明からのプログラム合成 (抽出)-続き

構成的プログラミング:

- 作成したいプログラム (関数) の仕様を書く。すなわち、入力変数 x と出力変数 y の間に成立する関係を論理式で記述する。
- この論理式を $\forall x.\exists y.\ M(x,y)$ の形にして、これを、直観主義論理で証明する。(これが、「プログラム」である。)
- 特定の x の値に対する y の値を知りたいとき、$\forall E$($\forall$ の除去規則) を、上記の証明に適用してから、「回り道の除去」を行う。(これが、「プログラムの実行」に相当する。)
- 回り道の除去が終了したとき、証明の一番下は $\exists I$($\exists$ の導入規則) ができているはずなので、y の値が具体的に書かれているはずである。

このようにして、得られた y の値が、$M(x,y)$ を満たすことの証明も同時に得られている。(100%正しいことが保証されたプログラムが合成できた!)

証明からのプログラム合成 (抽出)-まとめ

- 証明から、関数型プログラム言語のプログラムを「抽出」できる。
- 作成されたプログラムが、仕様を満たす (正しい) ことが保証される。
- 論理として直観主義論理を使うところが鍵。

型システムの拡張

単純型付きラムダ計算と命題論理の対応:

型	論理記号
$\times$	$\wedge$
$+$	$\vee$
$\rightarrow$	$\supset$

対応関係の拡張

型	論理記号
依存型 (Σ, Π)	一階述語論理 $(\forall^1, \exists^1)$
多相型 $(\forall)$	二階命題論理 $(\forall^2)$
コントロールオペレータの型	古典論理の論理式

型システムの拡張 1: 依存型

「長さ n のリスト」の型 $\mathtt{List}[n]$ が書けるとしたら：
$append([1;2],[3;4;5]) = [1;2;3;4;5]$ となる append 関数の型は、

$$append : \mathtt{List} \times \mathtt{List} \rightarrow \mathtt{List}$$
$$append : \mathtt{List}[n] \times \mathtt{List}[m] \rightarrow \mathtt{List}[n+m]$$

リストの長さを表すことにより、精密な型となる。

型システムの拡張 1: 依存型

もっと正確に：

$$append : \mathtt{List}[n] \times \mathtt{List}[m] \rightarrow \mathtt{List}[n+m]$$
$$append : \Pi n : \mathtt{Nat}.\ \Pi m : \mathtt{Nat}.(\mathtt{List}[n] \times \mathtt{List}[m] \rightarrow \mathtt{List}[n+m])$$

依存型 (dependent type):

- $\Pi x : A.\ B$　$x : A$ をもらって B 型のものを返す関数の型
- $\Sigma x : A.\ B$　$x : A$ と、B 型のものの対の型
- どちらのケースも、B に x を含んでもよい (B は x に依存)

応用例: buffer overflow 攻撃への対策: 配列のインデックスが、配列のサイズを越えた先を指さない、等の性質を、型システムにより保証。

型システムの拡張 1: 依存型と限量子

依存型 Π と限量子 $\forall$:

- $\Pi x : A.\ B$　$x : A$ をもらって B 型のものを返す関数の型
- $\forall x \in A.\ B$　「任意の $x \in A$ に対して、命題 B が正しい」という命題

依存型 Σ と限量子 $\exists$:

- $\Sigma x : A.\ B$　$x : A$ と、B 型のものの対の型
- $\exists x : A.\ B$　「ある $x \in A$ に対して、命題 B が正しい」という命題

まとめると

- 依存型〜限量子 (全称記号、存在記号)
- 依存型のある型システム〜一階述語論理

cf. Agda ‥依存型を持つ関数型プログラム言語

型システムの拡張 1: 依存型と限量子

構成的プログラミングの例で、既に、この関係を使っていた。

- 限量子と依存型の対応
- 数学的帰納法と再帰呼び出し (*) の対応

(*) 正確には、原始再帰法 (primitive recursion)

- 関数 $f : \mathtt{nat} \to A$。
- $f(0)$ の値を、A 型の要素として決める。
- $f(n+1)$ の値を、$f(n)$ の値を利用して決める (再帰)。

これは数学的帰納法と同じ構造。

型システムの拡張 2: 多相型

$\lambda x.x$ という (型のない) ラムダ項の型は？

- $(\lambda x : \mathtt{Int}.\ x)\ :\ \mathtt{Int} \to \mathtt{Int}$.
- $(\lambda x : \mathtt{String}.\ x)\ :\ \mathtt{String} \to \mathtt{String}$.
- $(\lambda x : \alpha.\ x)\ :\ \alpha \to \alpha$.

型システムの拡張 2: 多相型

多相型 (polymorphic type)

$$
\begin{aligned}
\mathit{double} &: \forall\alpha.\ ((\alpha \to \alpha) \to (\alpha \to \alpha)) \\
\mathit{double} &= (\lambda f : (\alpha \to \alpha).\ \lambda x : \alpha.\ f(f(x))). \\
\mathit{map} &: \forall\alpha.\forall\beta.((\alpha \to \beta) \times \mathtt{List}(\alpha) \to \mathtt{List}(\beta)) \\
\mathit{map}(\lambda x.\ x^2,\ &[1;2;3]) = [1;4;9] \\
\mathit{append} &: \forall\alpha.(\mathtt{List}(\alpha) \times \mathtt{List}(\alpha) \to \mathtt{List}(\alpha)) \\
\mathit{append}([1;2;3],&[4;5]) = [1;2;3;4;5]
\end{aligned}
$$

$\forall$ は、2 階命題論理の $\forall$ に対応する。
(1 階)$\forall^1 x.\forall^1 y.\ x + y = y + x$.
(2 階) $\forall^2 X.\forall^2 Y.\ (X \wedge Y) \supset X$.

型システムの拡張 3: Back to Classical Logic

- 直観主義論理 (計算のための論理): $A \vee \neg A$ や $(\neg\neg A) \supset A$ が不成立。
- 古典論理 (日常推論や数学のための論理)
- 直観主義論理が関数型プログラム言語 (型付きラムダ計算) に対応することはわかったとして、それでも、古典論理に対応する計算体系 (プログラム言語) はないのだろうか？

型システムの拡張 3: Back to Classical Logic

コントロールオペレータ:

- C の longjmp/setjmp
- Java の try-catch-finally
- ML の exception (例外)
- Lisp の catch/throw
- Scheme の call/cc
- 制御の流れを変更する構文要素。

型システムの拡張 3: Back to Classical Logic

拡張された Curry-Howard の同型対応: 型付きラムダ計算にある種のコントロールオペレータを追加すると古典論理に対応する!!
バックトラックありの証明:

- $A \vee \neg A$ の証明 (プログラム) は?
- $inr(M)$ の形をしている ($\neg A$ の証明であるような顔をしている)。
- もし、その後の計算において、M が使われるならば、($\neg A$ の証明であるので) 当然 $M(N)$ の形である。ここで N は A の証明。
- よって、この時点で A が正しかったことがわかるので、バックトラックして、$A \vee \neg A$ の証明に戻り、$inl(N)$ を証明として提供する。

型システムの拡張 3: Back to Classical Logic

- call/cc や exception を利用して、実際に $A \vee \neg A$ の証明を書くことができる。
- これにより、数学の本に載っている証明 (ほとんどすべて古典論理を使って記述される) に対応するプログラムが何であるかわかった。
- ただし、一度正しいと思ったことが、後でひっくり返される可能性のある「恐い」証明ゲームである。
- 「回り道の除去」により、call/cc や exception が全部消滅したら、「普通の証明」が得られる。

授業のまとめ 1.

- 型付きラムダ計算におけるプログラム言語の厳密な取り扱いの基礎を学んだ。
- 構文と意味論の両方が、形式的な演繹の形で与えられる (CAL ゲーム)。
- ラムダ計算における値呼び戦略と名前呼び戦略を学んだ。
- プログラム言語と論理の密接な関係を見た。

授業のまとめ 2.

皆さんがもともと (この授業の前から) 知っていたこと：

- ラムダ計算の動的側面: 計算 (関数型プログラムの実行)
- 命題論理の静的側面: 証明可能性

この授業で新たに学んだこと：

- 型付きラムダ計算の静的側面: 型システム
- 直観主義命題論理の動的側面: 証明 (導出) の回り道の除去 (ある種の計算)
- Curry-Howard の同型対応
- 型推論

授業のまとめ 3.

皆さんが将来学ぶかもしれないこと：

- 型システムを使ったプログラムの静的解析
  例: Java ByteCode Verifier は、型推論を自動的におこなうことにより、実行時に型エラーが起きないこと、それぞれのメソッドがスタックを無限に消費しないこと、ジャンプの飛び先が必ずプログラム内に存在すること、などを保証。
- 様々な論理体系における自動証明手続き
  例: SAT-solver: 古典命題論理に対する非常に高速な自動証明器
  (ハードウェア設計、システム検証などをはじめとして、極めて色々な分野で利用されている。)