In Laboratory Now

研究室訪問 5

λ計算の可能性を追求する

西崎 真也 研究室～計算工学専攻

西崎 真也 准教授

コンピュータプログラムの最適化や安全性を高めるという目的で、プログラムがどのような構造を持ち、動作をしているのかということを解析する必要がでてきた。この解析をするために、プログラミング言語がどのような構造を持つのかを知る必要がある。

西崎研究室では、プログラミング言語を解析するための基礎を研究している。プログラミング言語の文法や、文法に従って記述されたプログラムの意味の定義の方法を定義することにより、プログラムの動作を解析することが可能になる。

コンピュータプログラムの解釈

我々は日常的に言語を用いて考え、表現する。言語は、自然言語と形式言語の二つに分けることができる。日本語や英語、ドイツ語などが自然言語に当たり、日常生活で用いる言語である。形式言語は数式や化学式のようなものを指す。

西崎研究室では、コンピュータの動作を記述するための形式言語であるプログラミング言語を研究している。コンピュータの一連の動作の指示をプログラムと呼ぶ。プログラムは、コンピュータが理解できるコンピュータ内部の電気信号の流れを指示する形で書くこともできるが、通常そのように書くことはない。プログラムの記述はある程度人間にわかりやすく、数値の加算、乗算というように抽象化された形でなされる。その抽象化された形で記述するための言語がプログラミング言語である。言語で記述されるという点では、プログラムはコンピュータにさせたい動作を記述した文章のように捉えることができる。

プログラミング言語で記述されたプログラムは抽象化された形で記述されており、コンピュータはそのままでは、指示として読み取ることができない。そこで､何らかの方法によって､コンピュータが理解できる形に変換する必要がある。変換されたプログラムもまた、プログラムと呼ばれる。変換は「解釈器」が行う。解釈器はプログラミング言語で記述された文章を読み取り、その各要素がどのような動作をコンピュータに指示するかを解釈して、コンピュータが理解できる形に変換する。すなわち、解釈器は人間とコンピュータの通訳の様な役割を果たす。解釈器もプログラムであり、通常プログラミング言語で記述される。プログラミング言語で記述された解釈器を、コンピュータが理解できる形に変換するのもまた解釈器であり、元をたどると、コンピュータが理解できる形で直接書いた解釈器が存在する。

解釈をする中で、プログラミング言語がもつ文法や、プログラム中の要素と動作の対応関係をどのように定義するのかという問題が重要となる。

西崎研究室で行われている研究を紹介するにあたり、まず、プログラミング言語で書かれたプログラムと動作の対応を解析するための一般的手法を説明する。次に、西崎研究室で行われている、プログラム中で解釈器の動作を変更する指示をプログラミング言語に導入する研究を紹介する。

構文論と意味論

言語の構造

プログラミング言語は形式言語の一種であり、その文法は、形式言語一般では構文論と呼ばれ、コンピュータの動作との対応関係は、形式言語一般の意味論に当たる。古典物理学における運動方程式を例にして、構文論と意味論の位置づけを説明する。

古典物理学での目標は、ある時点での物体の状態から、物体の動きを予測することである。その目標を達成するために、まず、座標系を設定し、文字と物理量を結びつける。次に、運動方程式と呼ばれる数式を立てる。すると、その数式を変形し、はじめに設定した座標系や文字と物理量の対応に従って解釈することで、物体の運動の予測をすることができる。これらの流れの中で用いられる言語において、数式の組み立て方とその変形方法が構文論で定義され、座標の設定の仕方や文字の置き方の規定が意味論で定義される。これら一連の流れには**図 1** のような関係が成立している。

数式もまた形式言語の一つであるため、構文論と意味論が存在する。数式が満たすべき文法的なルールを構文論と呼び、構文論以外の、文字列と問題との関連は意味論と呼ぶ。意味論はモデルと解釈という二つの要素から構成される。数式の要素の定義をモデルと呼び、数式とモデルを仲介する部分を解釈と呼ぶ。**図 1** における問題の部分を、形式言語一般では意味と呼ぶ。意味論は、構文論に従っている文字列と、それが実際に意味しているものとの対応関係を規定している。

形式言語を構文論と意味論の二つに分離することによって、文字列が意味するものから独立して研究することができる。古典物理学における運動方程式を解くことにより、物体の運動が予測できるという理論は、これら数式の構文論と数式の意味論に加え、モデルが物理的な事象を表しているという制約の上で成立する。この理論が一般的に成立するのは、細かい物理的な事象が意味論によって覆い隠されているからだ。構文論と意味論に分離する利点はそこにある。

プログラミング言語も、他の形式言語と同様に、構文論と意味論という視点から研究することができる。プログラミング言語の形式言語としての側面をみることで、コンピュータプログラムがどのような問題を扱えるのか、言語の特性はどのようなものかということが研究できる。しかしながら、プログラミング言語はその複雑さゆえ、直接扱うのは非常に難しい。そのため、プログラミング言語の研究では、プログラミング言語を抽象化した言語である λ 計算を扱うことが多い。西崎研究室でも実際のプログラミング言語ではなく、λ 計算を用いて研究を行っている。

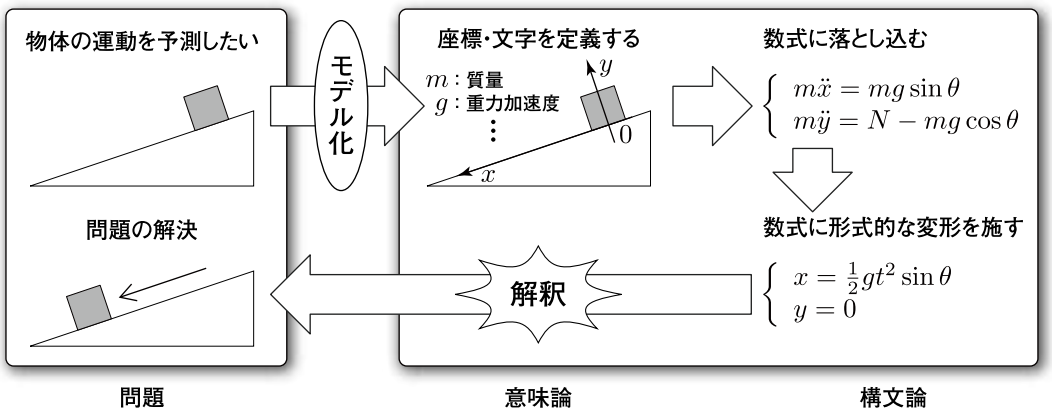


図 1　古典物理学における構文論と意味論

λ計算の構文論

西崎研究室の研究について説明するために、λ計算の概略を述べる。

数式が数を扱う言語と言えるのと同じように、λ計算は「写像を扱う言語」と言える。写像とは「集合の要素と要素の対応関係」である。数学における関数は写像の一種であり、例えば「$f(x)$=x+2」という関数は、実数から実数に写す写像である。このことを実数の集合 $\boldsymbol{R}$ を用いて「f:$\boldsymbol{R}\rightarrow\boldsymbol{R}$」と書く。また「$f(x)$=$x$+2」という写像に対して、「$f(2)$」のように、写したい要素を指定することを適用と呼ぶ。この場合、「2に f を適用する」と言い、「$f(2)$=2+2=4」となるので、適用した結果「4」を得ることになる。

プログラミング言語でプログラムを記述するときに、その記述の大部分が関数の操作で占められるため、写像を表す言語であるλ計算とプログラムは非常に親和性がよい。実際に、プログラムの記述に使われているプログラミング言語には、λ計算を基礎理論とするものは多い。

λ計算の構文論に従っている式をλ式と呼ぶ。λ式における表記は**表1**のようにする。

表1 (1) は写像の表記方法である。「$\lambda x.$」の部分が「$f(x)$」の部分と対応してパラメータを表している。「.」以降の部分の説明は後述する。名前の上では別の写像であっても、写像自体が同じであれば、λ式での表記は同じになる。例えば**表1 (1)(2)** のように、関数 f と g は数学における表記が異なるが、λ式では表記が同じになる。**表1 (3)** は写像の適用を表している。10を f に適用することを、数学では「$f(10)$」と表すのに対して、λ式では「$(f\ 10)$」と表す。**表1 (4)** で示すように、適用したい写像をその場で記述して、適用させるように表現することもできる。

また、次のような定義で、数をλ式として扱うことができる。

$$0 := (\lambda f.(\lambda x.x))$$
$$1 := (\lambda f.(\lambda x.(fx)))$$
$$2 := (\lambda f.(\lambda x.(f(fx))))$$

この定義を用いることにより、写像も値もλ式として表現できる。つまり、**表1 (3)(4)** は(写像 値)の形を取っているが、これは(λ式 λ式)という形式だと言い換えることができる。

	数学における表記	λ記法による表記
(1)	$f(x) = x + 2$	$(\lambda x.(+\ x\ 2))$
(2)	$g(x) = x + 2$	$(\lambda x.(+\ x\ 2))$
(3)	$f(10)$	$(f\ 10)$
(4)	$x + 2\|_{x=10}$	$((\lambda x.(+\ x\ 2))\ 10)$

表1　λ式の例

以上がλ式の構文である。「$f(x)$」の「x」や、「$f(10)$」の「10」のことは引数と呼ばれる。特に前者のようなパラメータとしての引数を「仮引数」、後者のような具体的な値としての引数を「実引数」と呼んで区別する。λ計算の記法の利点の一つとして、数学では区別されにくい仮引数と実引数を明確に区別できることが挙げられる。例えば、数学では単に「$f(y)$」と書いたとき、これが「y を仮引数としてとる写像」なのか「$f(x)$ に対して y という実引数を与えた」のかの区別がつかない。λ計算の記法では「$\lambda y.$」という形で前者を示し、「$(f\ y)$」で後者を示すというように、区別ができるようになっている。

写像の引数が一つの場合、前述したとおりに表現できる。これが二つ以上の場合も、記述の仕方を工夫して**表2**のように表現できる。

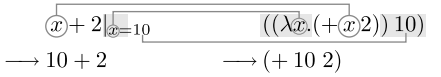
表2 (1) では $\boldsymbol{R}^2\rightarrow\boldsymbol{R}$ となる関数を、λ計算では $\boldsymbol{R}\rightarrow[\boldsymbol{R}\rightarrow\boldsymbol{R}]$、すなわち「ある実数を引数としてとり、『実数から実数へ写す写像』へ写す」写像として表す。まず、引数を一つ適用し、写像が得られる。次に、その写像に対してもう一つの引数を適用するという形になる。また、**表2**の中にあるように、それぞれ引数が多い場合は簡単な表記もすることができる。**表1 (1)** の「.」以降は、**表2 (2)** のような引数が多い場合の写像の適用を表している。

λ計算による表現で、特に**表1 (4)** のような形の場合、仮引数を実引数で置き換えることで、よ

	数学における表記	λ記法による表記	
(1)	$f(x, y) = x + y$	$(\lambda x.(\lambda y.(+\ x\ y)))$	
		$(\lambda xy.(+\ x\ y))$	ともできる
(2)	$f(10, 5)$	$((f\ 10)\ 5)$	
		$(f\ 10\ 5)$	ともできる

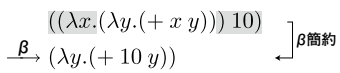
表2　引数が多い場合のλ式

り簡単な形にすることができる。この簡単にする操作を β 簡約と呼び、次のように表現できる。



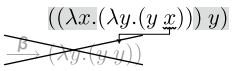
この β 簡約という操作は、写像の仮引数として指定された変数を、写像の中で探し、その変数を実引数で置き換えるという操作である。このように簡約した後、写像の「$\lambda x.$」は消去される。また、それに対応して実引数も消去される。この簡約の操作の中で、写像の形式は変化したが、写像が要素を写す先は変化していない。

β 簡約をすることで、引数が 2 つ以上の場合における写像の適用のなかで、新しく写像が生まれることがわかる。

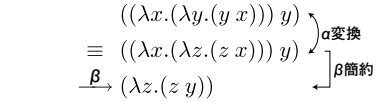


引数を一つのみ適用させると、簡約した結果がまた写像になる。簡約した結果の写像にもう一つの引数を適用することで、2 つの引数を取る写像が表現できる。

また、β 簡約をするときに、次のように仮引数が被ってしまうことがある。



そのまま β 簡約を行うと、実引数としての「y」だったものが、パラメータとしての「y」に変わってしまう。そこで、仮引数の変数を他の変数に置き換える α 変換を行う。



α 変換を行うことで、実引数と仮引数が被らないように変換することができる。β 簡約とは違い、写像の形式も写像が要素を写す先も変化していないことがわかる。

ここまで説明した λ 計算の記法を λ 式と言う。λ 式は次に表す再帰的な集合 E の要素としてまとめられる。

(1) 変数 $x, y, z, \ldots \in E$

(2) E の任意の要素 M について $(\lambda x.M) \in E$

(3) E の任意の要素 M, N について $(MN) \in E$

数や演算なども λ 式で表現できることは前に述べた。(1) はパラメータである変数も λ 式であるということを表している。(2) は、λ 式での写像が「ある λ 式の引数を取り、λ 式を返す写像」であることから、任意の λ 式 M について、そのパラメータを表す「$\lambda x.$」をつければ、写像を構成できることを表している。(3) は **表 1 (3)(4)** で述べたように、λ 式を二つ並べて括弧でくくることで、写像の適用を表すということである。

λ 式と実際の写像の関係は、後述する意味論で定義される。これらの構文論は、λ 式というかたちで写像を表現する方法と、意味論的におよそ妥当と考えられる変形を λ 式に施すルールとなる。意味論と構文論で λ 計算が構成できる。

λ計算の意味論

形式言語の構文論の例としてはじめに古典物理学の問題を数式に落とし込む例を挙げた。しかし、数式自体は単に記号を並べただけのもので、それに物理学上の意味はなかった。形式言語に意味論を導入し、言語に持たせたい具体的な意味のセットであるモデルを構成することによって、構文論の規則に従って構成される文字列と意味の関係が表される。

形式言語において、構文論は文字列の構成法と、その形式的な操作の定義であった。しかしながら、そのモデルがどうあるべきかを想定しなければ、言語として意味のあるものにならない。意味論はそういった、モデルが満たすべき条件を定義している。

モデルが満たすべき条件が存在することで、その条件を満たすと仮定し、言語の上で理論を形成することができる。このような理論は物理学における運動方程式が例としてあげられる。地球上での物体のモデルも、月上での物体のモデルも、モデルが適切に構成されていれば、運動方程式は成立する。運動方程式はモデルが地球上のものか、月上のものかという、具体的な詳細を仮定しない、一般的な理論である。このように、意味論がモデルの満たすべき条件を定義していることにより、

その条件に従って理論を構築できるのである。構文論と意味論の二つに分離すると、意味と構文論の間が区切れ、**図２**のように抽象の壁が生じる。この抽象の壁が個々のモデルの具体的な詳細を覆い隠している。

モデルが満たすべき条件を利用して、モデルという型にはめられた意味を文字列と結びつける写像が構成できる。この文字列とモデルの対応を表す写像を解釈と呼ぶ。

λ計算のモデルはλモデルと呼ばれる。λ計算の意味論となる、λモデルが満たすべき条件とそれに対応した解釈を説明する。

数といったものも写像で表せることから、すべてのλ式は次のようにEからEの写像に変換したり、逆にEからEの写像からλ式に変換することができる。

$$E \to [E \to E]$$
$$[E \to E] \to E$$

λ式のこの特徴と対応して、λ式の意味の集合Dについても、同様の特徴を持つ。例えば、数式の変数や数値といった要素は足したり掛けたりできるものである。それら要素の意味も、足したり掛けたりできるものでなければ、意味をなさない。物理学では、それぞれの要素に対し、長さや時間などの単位を設定して、次元をあわせることで、数式全体の意味を形成している。λモデルで上に示したλ式の特徴を表現するために、次のような写像が定義できる必要がある。

$$\varphi : D \to [D \to D]$$
$$\psi : [D \to D] \to D$$

意味の集合D､φ､ψをまとめてλモデルとなる。λモデルを利用して、λ式の集合Eと意味の集合Dを結びつける、解釈〚 〛が構成できる。

解釈はλ式の集合Eから、λ式の意味の集合

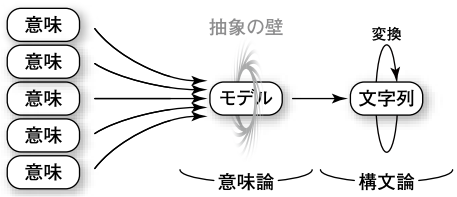


図２　モデルと抽象の壁

Dへの写像である。この解釈によって、λ式とその意味を結びつける。構文論の説明で触れた、λ式の再帰的な集合を基にして、解釈の構成方法を述べる。

まず、「$(\lambda x.M)$」はEからEの次のような写像であると考えられる。

$$f(x) = M \quad (f : E \to E)$$

この写像は、引数としてλ式を受け取って、また別のλ式を返すものである。これと対応して、次のように意味の集合DからDへ写す写像が考えられる。

$$F(d) = [\![M]\!] \quad (F : D \to D)$$

この写像は、引数として受け取った意味を別の意味に変化させている。

ここでは、Eの要素である引数「x」とDの要素「d」が対応している。しかしその対応は、λ式「M」の解釈の中まで伝えられず、このままでは無視されてしまう。「M」中の変数「x」の解釈が「d」に割り当てられるようにするため、環境という概念を導入する。

環境は[変数の集合→ D]となる写像で、λ式における写像について意味の割り当てを行う写像であり、「$\rho(x)$」と表す。先の「x」に対して「d」を割り当てるとき、環境のなかで「x」を写す先を「d」にする必要がある。これを「$\rho[x \mapsto d]$」と表現する。この環境から「d」を取り出すには、「$\rho[x \mapsto d](x)$」のように、引数として「x」を渡せばよい。

この環境を利用して、先ほどのFは次のように変形できる。

$$F(d) = [\![M]\!]_{\rho[x \mapsto d]} (F : D \to D)$$

このように､解釈で用いる環境を共に明示する。

λモデルの中のψを利用することにより、このDからDの写像FはDに変換することができる。変数単体の解釈は環境からDの要素を取り出すとすることができる。「$(M\ N)$」の解釈は、まず、「M」の解釈した結果に対して、λモデルのφを適用してDからDの写像に変換する。この写像を「N」を解釈した結果に適用することにより、Dの要素を得ることができる。

このような推論より、λ式の解釈は次のように

再帰的に構成できることがわかる。

λ式を解釈した結果に出てくるλ式は、単に形

(1) $[\![x]\!]=(\lambda\rho.\rho(x))$

(2) $[\![(\lambda x.M)]\!]=(\lambda\rho.\psi(f))\quad \left(f{:}D\to D, d\mapsto [\![M]\!]_{\rho(x\mapsto d)}\right)$

(3) $[\![(MN)]\!]=(\lambda\rho.(\varphi([\![M]\!]_\rho))([\![N]\!]_\rho))$

式的に引数を表現するもので、厳密な意味でのλ式ではないことに注意してもらいたい。

解釈はλ式と環境を受け取り、意味の集合Dの要素を返す。このようにすることによって、環境に解釈を適用するような形にできる。(1)は変数について環境から意味を取り出すように定義している。(2)は、まず、意味の集合Dの要素を一つとり、それを変数xに割り当てた環境で「M」を解釈した結果を返す関数fをつくる。その$f{:}D\to D$をψでDに変換し、それを解釈した結果とするということである。(3)は「M」を解釈した結果をφで$D\to D$の関数に変換し、その関数を「N」を解釈した結果に適用する。その結果として、Dの要素を得るので、それを「$(M\ N)$」の解釈の結果とするということである。

「$(\lambda x.M)$」の形式のλ式を解釈をする中で、逐一関数を定義すると煩雑になってしまうため、次のようにλ式の形式を用いて簡単に表記する。

$$[\![(\lambda x.M)]\!]_\rho = \psi(\lambda d.[\![M]\!]_{\rho(x\mapsto d)})$$

例えば、「$(\lambda f.(\lambda x.(f\ x)))$」を解釈してみると、以下のようになる。

$$\begin{aligned}
&[\![(\lambda f.(\lambda x.(fx)))]\!]_\rho \\
&= \psi\left((\lambda d_1.[\![(\lambda x.fx)]\!]_{\rho[f\mapsto d_1]})\right) \\
&= \psi\left((\lambda d_1.\psi\left((\lambda d_2.[\![(fx)]\!]_{\rho[f\mapsto d_1, x\mapsto d_2]}\right)\right)\right) \\
&= \psi\left((\lambda d_1.\psi\left((\lambda d_2.(\varphi([\![f]\!]_\rho{\scriptstyle\left[\substack{x\mapsto d_2\\ f\mapsto d_1}\right]}))([\![x]\!]_\rho{\scriptstyle\left[\substack{x\mapsto d_2\\ f\mapsto d_1}\right]})\right)\right)\right) \\
&= \psi\left((\lambda d_1.\psi\left((\lambda d_2.(\varphi\,(d_1))\,(d_2)\right)\right)\right)
\end{aligned}$$

このようにして、意味と文字列の間に対応づけを構成することができる。

環境をファーストクラスへ

λ計算について簡単な概略を述べた。構文論と意味論という二つの立場をとることにより、その言語における表現・操作と、言語の意味との対応関係を分離した。

構文論と意味論で定義された、規則やモデルの条件、解釈の構成方法を変更することで、λ計算を拡張することができる。西崎研究室では拡張されたλ計算を提唱し、さらにそのλ計算における命題の成否がどのように変化するのかを研究している。現在は、解釈を定義する過程で用いた環境をλ計算上で扱えるようにする研究を行っている。拡張前のλ計算で扱えるのはλ式のみであったが、この拡張によって環境も扱えるようにすることができる。このように、ある言語において扱うことのできるものを、「ファーストクラスオブジェクト」と呼ぶ。

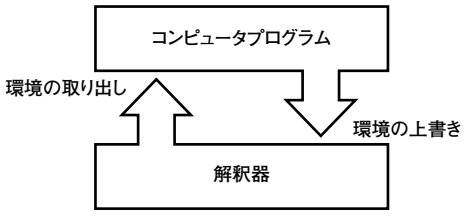


図3　解釈器から環境を取り出す

西崎先生は環境をファーストクラスオブジェクトとして扱うようにするために、次のような規則を付け加えた。

$$\begin{aligned}
[\![(\text{the-environment})]\!] &= (\lambda\rho.\rho) \\
[\![(\text{eval }`MN)]\!] &= (\lambda\rho.[\![M]\!]([\![N]\!]_\rho)) \\
[\![(M/x)\cdot N]\!] &= (\lambda\rho.[\![N]\!]_{\rho([x\mapsto[\![M]\!]_\rho])})
\end{aligned}$$

「(the-environment)」という記号は、その記号を解釈した時点での環境として解釈される。また「(eval ‘M N)」は、まず環境Nを解釈し、その解釈された環境Nで、「M」という記号列を解釈する。これは意味論で定義した解釈の方法により、「M」の解釈を「N」の解釈を適用するという形で簡単に書ける。「$(M/x)\cdot N$」は、現在の環境に「x」という変数を$[\![M]\!]_\rho$の割り当てを加えた環境を作り、その環境でNを解釈するということである。

今まで、環境は、λ計算を解釈するシステム、解釈器のみで扱っていた。この環境を、解釈されるλ式自体が扱えるようになることで、λ式自身が自身の意味を書き換えることが可能になる。つ

まり、λ計算が解釈器に近い能力を持つことになるのだ。**図3**で示すとおり、環境を扱えるようにするということは、構文論に意味論を書き換える操作を許すことに他ならない。

しかし、このような拡張をすることによって、λ計算としての性質が壊れてしまうことがある。例えば、次のような場合が考えられるため、α変換によって変換された式の解釈が、常に同じであるとは言えなくなってしまっている。

$$((\lambda x.(\text{the-environment}))1) \Rightarrow \rho[x \mapsto 1]$$
$$((\lambda y.(\text{the-environment}))1) \Rightarrow \rho[y \mapsto 1]$$

左側の式は、互いにα変換をすることで得られるが、二つの式を解釈した結果は違うものになっている。他方では、拡張前のλ計算では成立しなかった性質をもつこともある。強正規化可能性定理などはその代表例だ。

西崎先生は、λ計算で環境をファーストクラスオブジェクトとして扱えるように拡張を行い、その拡張したλ計算で、先に出した強正規化可能性定理のような基本的な定理を証明した。

本稿で構文論と意味論を紹介したλ計算は「型無しλ計算」と呼ばれる。他にも、型という概念を付け加えた「型付きλ計算」、「単純型付きλ計算」、「二階型付きλ計算」、「多相型付きλ計算」があり、西崎研究室ではこれらについても、同様の研究を行っている。

これら型付き・型無しのλ計算は、それ自体に数学的な対応が存在することがわかっている。特に型付きλ計算の型システムが、古典論理や直観主義論理と対応することが知られている。単純型付きλ計算の型システムは、直観主義論理と対応することがわかっており、このλ計算に「継続」という概念をつけることで、古典論理と対応するようになる。これは継続そのものが、Pierce's law と呼ばれる「$((A \to B) \to A) \to A$」と対応していることによる。西崎先生は、継続に数学的対応が存在するのと同様に、環境にも数学的対応がないかという動機で、この研究を始めた。

環境をファーストクラスにするという概念は、元々、λ計算を基礎理論とするプログラミング言語 Scheme に存在している。環境をファーストクラスオブジェクトとして扱う言語は他にもあり、例えばプログラミング言語 Ruby の binding オブジェクトはまさに環境そのものとなっている。

環境がファーストクラスオブジェクトとして扱えるプログラミング言語は、まだそれほど多くはない。また、扱えるようになっていても積極的に使われているとは言えない。環境は言語の意味論に密接に関わることのある概念のため、言語の可能性を広げる代わりに、扱いが非常に難しくなってしまう。だが、環境を書き換えることができることで、既にある言語の構文論を用いながら、全く異なる言語を作ることが可能になる。プログラミング言語を新たに作るときに、文法は既にあるプログラミング言語のものを利用しつつ、解釈したときの動作のみ変えるということを、環境を用いてプログラムの上で記述できる。このため、言語を最初から設計するよりも大きく手間が省けることになる。

環境の性質・機能の一部を扱えるようにして言語の柔軟性を上げている言語もある。例えば、自己反映計算は、環境の一部を書き換えられるようにしたものに他ならない。環境は名前を変えて、いろいろな言語で扱えるようになりつつある。

最近、プログラミング言語自体が定理の証明に使われたり、プログラムの記述と同時に、プログラムが、目的の動作をしていることの証明を同時に行われたりしている。このとき、プログラミング言語が整合性がとれているかどうかが重要になってくる。プログラミング言語の整合性の確認にも、形式言語としての解析が必要になる。

環境が徐々に採用されていく状況の中、西崎研究室の研究によって、プログラミング言語に関する理論が整い、その理論によって、言語の可能性を広げることができることを切に願う。

私個人の強い興味から、このたび西崎研究室を取材させていただきました。普段、λ計算を基礎理論とした言語に触れることはあっても、なかなかその理論自体に触れることはありません。記事を執筆するにあたり、そのような勉強する機会を与えてくださり、取材を快諾してくださった西崎先生に感謝いたします。

（鈴木　将哉）