計算と論理のための自然枠組 NF/CAL

佐藤 雅彦

本論文では，計算機上での証明記述および検証を支援するシステム NF/CAL を紹介する．NF/CAL の設計はフレーゲによる判断の分析の影響を受けており，人間が自然言語を用いて日常的に行う判断活動を自然に形式化したものになっている．

This paper gives a survey of the software system NF/CAL which supports both development and verification of formal proofs on a computer. The NF/CAL has been designed under the influence of Frege's analysis of judgments, and as a result, it naturally formalizes our act of judgment which we perform daily using natural languages.

1 はじめに

本稿では，筆者の研究室で 1998 年から開発を続けているソフトウェアシステム NF/CAL について，その概要と哲学的背景を述べる．NF は Natural Framework (自然枠組) の略称であり，CAL は Computation and Logic (計算と論理) の略称である．

NF/CAL は 1998 年に京都大学工学部情報学科に開設された講義科目「計算と論理」のための演習用システムとして開発された．NF/CAL は教育用システムであるだけでなく，ソフトウェアの正しさの検証や，仕様から正しさの保証されたソフトウェアを構築するための環境を提供することも目指している．

本来，数理論理学は数学等で人間が日常的に用いている論理を形式化し体系化したものを研究する学問である．しかしながら，現在論理の教育において採用されている様々な形式体系は，いずれもその体系の中で具体的な証明を構築することは極めて困難なものが多い．その理由は，数理論理学はメタ数学 (超数学ともいう) とよばれる数学の分野であり，そこでは数学自身を形式化により考察の対象とし，形式化された体系を外から観察し分析することが一般的であるからである．したがって，数理論理学の研究において個々の形式的な証明を具体的に構築することは稀であり，そのような証明の存在の可能性についての議論が主である．そのため，体系の外からの扱いやすが重視され，体系の中での活動の容易さは軽視されるという傾向が続いている．

計算の理論についても同様の事情がある．1930 年代の初頭に計算可能性の概念が定式化されたが，そこで提案された代表的な計算モデルのひとつにチューリング機械がある．チューリング機械は，原理的には，現在実際に使用されている様々なプログラミング言語と同等の計算能力がある．しかしながら，チューリング機械自体をプログラミング言語として使うということは，練習問題を別にすれば，実用的でないために行われていない．ここでも，チューリング機械は計算可能な関数や計算可能でない関数についての一般論を展開するための理論的な道具としての有用性が重視されていることを見ることができる．

計算や論理といった情報学を学ぶ上での基礎となる事項の学習においては，上で述べたような理論的装置としては優れていても実用上に問題のある形式

A Natural Framework for Computation and Logic: NF/CAL.

Masahiko SATO, 京都大学大学院情報学研究科, Graduate School of Informatics, Kyoto University.

コンピュータソフトウェア, Vol.00, No.0 (2006), pp.1–11.
[チュートリアル] 2006 年 yy 月 zz 日受付.

体系を排除し，理論的にも優れていて同時に実用的な形式体系を取り上げることが重要であると考えた．「計算と論理」の講義では，講義で扱うすべての形式的体系を計算機上にも実装し，学生が実際にこれらの体系の中で遊ぶことができる環境を提供し学習の助けとすることを目標にした．この目標を実現したシステムが NF/CAL である．

NF は計算と論理に関する様々な体系の記述を可能にする極めて一般的な枠組であり，NF/CAL システムの理論的基盤を与えるものである．NF で記述される個々の体系は導出ゲームとよばれ，NF はこれらの導出ゲームに対するメタシステムとなっている．理論体系 NF を計算機上に実装し，同時に NF とユーザとのインターフェースを提供するシステムが CAL である．

以下では 2 節で自然枠組を下から支え，その形式的表現手段を提供する式の理論を説明し，3 節で自然枠組を解説する．4 節では CAL の概要を述べ，5 節で結語と今後の展望を与える．

2 式

最初に NF/CAL システムは以下の図 1 に示す階層構造をもつことを注意しておきたい． ここで，上

対象と概念 (導出ゲーム)
判断と導出 (NF)
式 (CAL)
S 式 (Emacs Lisp)

図 1 NF/CAL の階層構造

位の階層はその直下の階層により実現されるという関係がある．最下位の階層はプログラミング言語 Lisp の一種である Emacs Lisp により提供される階層であり，CAL システムは Eamcs Lisp を用いて実装されている．Emacs Lisp を採用した理由は，計算機上でテキストを扱うための強力な環境である Emacs 上で自由に利用できるため CAL 上にユーザインターフェースを容易に構築できるということと Lisp が提供する S 式をデータ型として利用できるからである．また，このような階層化は，直接の上下関係にある階層の間にのみインターフェースを実現すればよく，さらに下位の階層を抽象化して実装することができるという利点を生む．

本節で解説する式 (expression) は，具体的には Emacs Lisp が提供する S 式を用いて実現されるデータであるが，ここでは S 式の概念には依存せず直接的な定義を与える．式の理論や，式の上位階層が取り扱う判断と導出の理論は，「計算と論理」を開講したときの CAL システムにはなかったものであるが講義の経験を通じて自然発生的に必要な概念として獲得されたものである．

次節以降で詳しく議論するが，NF の根幹をなす基本概念は判断 (judgment) と導出 (derivation) である．ここで，判断も導出も言語により表現可能であり，人間同士さらには人間とコンピュータ，コンピュータ同士で伝達可能であるという特徴がある．このような言語表現を統一的に実現するデータ構造がこれから説明する式である．

McCarthy が定義した Lisp の S 式 [8] はアトムからはじめて，対をつくる操作 (cons) を繰り返し適用することにより得られるデータ構造である．S 式の全体はアトムの全体を生成元とする自由マグマとなる．ここでマグマ (magma) は Bourbaki による用語であり 2 項演算をひとつもつ代数系のことである．以下に定義する式は，S 式にある対をつくる操作にさらに抽象体 (abstract) をつくる操作を追加してできたものと考えられる．

式はその上位層にある判断と導出を表現する形式を与えるものである．ここで，判断は日常語の意味で考えてよいが，よりはっきりさせるためにここでは数学的な判断を具体例として考えることにする．数学的な判断は，その正しさを主張するためには，正しさの根拠として証明が求められる．証明には間違いがあることもあるが，レフェリー制度のような社会的なプロセスを通じて主張が受け入れられたものは定理として認知される．このようなプロセスを支えているのが，主張をする人とそれを受け入れる (または否定する) 人との間の言語によるコミュニケーションである．すなわち，判断およびその根拠としての証明は

言語化可能であり伝達可能であるという特徴がある．証明は判断の根拠を導き出すという行為の結果を言語化したものなので，自然枠組では証明のことを導出 (derivation) と呼んでいる．

判断はその正しさを主張する文として表現される．たとえば，「ソクラテスは人間である」は判断である．フレーゲ [5] は，概念や関係を (高階の) 関数としてとらえて，文はそのような関数を引数に適用することにより得られると考えた．そして，フレーゲは，関数自身は不飽和 (unsaturated) であり，引数を補充することにより飽和した (saturated) 文が得られるとした．例えば，上の文は不飽和な関数である概念「□は人間である」の空所 (□) を引数「ソクラテス」で補充することによりできていると分析することができる．

以下ではアリティ (arity) とカテゴリー (category) の概念を導入し，フレーゲによる判断の分析を取り入れつつ S 式の拡張として式を定義する．

2.1 カテゴリー

すべての式はただひとつのカテゴリーに属するように定義される．式 a がカテゴリー κ に属するという判断を $a:\kappa$ で表す．カテゴリーは有限個の基底カテゴリー (ground category) を指定することにより決定される．基底カテゴリーは，式の属する構文的カテゴリー (syntactic category) を定めるものである．本稿では，話を具体的にするために，判断 (judgment) を表すカテゴリー j，対象 (object) を表すカテゴリー o，導出 (derivation) を表すカテゴリー d の 3 つの基底カテゴリーを考える．基底カテゴリーを γ，一般のカテゴリーを κ で表すことにし，これらのカテゴリーを以下の文法で定める．

$$\gamma \quad ::= \quad \mathsf{o} \mid \mathsf{j} \mid \mathsf{d}$$
$$\kappa \quad ::= \quad \gamma \mid (\kappa_1,\ldots,\kappa_n)\gamma \ (n>0)$$

基底カテゴリー κ を一階のカテゴリーとも言い，$(\kappa_1,\ldots,\kappa_n)\gamma\ (n>0)$ の形のカテゴリーを高階のカテゴリーと言う．

2.2 アリティと式

式は変項 (variable) と定項 (constant) を組み合わせて構築される．各変項および定項には一意的にカテゴリーを割り当てる．これにより，Lisp の S 式のように式を自由に組み合わせることは禁止し，一定の文法規則にしたがう表現だけを式として構築することができるようになる．

変項や定項に割り当てられたカテゴリーのことをアリティ (arity) と言い，変項 x (定項 c) にカテゴリー κ が割り当てられているとき x (c) はアリティ κ を持つと言って $x@\kappa$ ($c@\kappa$) と書く．アリティ κ を持つ変項や定項に対して，そのアリティ数 (arity number) を次のように定める．κ が基底カテゴリーのときには 0，$\kappa=(\kappa_1,\ldots,\kappa_n)\gamma\ (n>0)$ のときには n. アリティ数はその変項や定項がとる引数の数を表すものである[†1]．カテゴリー $(\kappa_1,\ldots,\kappa_n)\gamma$ をアリティとみなすときには $\gamma[\kappa_1,\ldots,\kappa_n]$ と書くことにする．

変項と定項には重複がなく，また必要なだけいくらでもあるとし，さらに各変項，定項にはアリティが一意的に定まっているとする．このとき式は以下のように定義される．各式 a にはそれが属するカテゴリー κ が一意的に対応し，このことを表すメタ判断を $a:\kappa$ と書く．

$$\frac{x@\gamma[\kappa_1,\ldots,\kappa_n] \quad a_1:\kappa_1 \quad \cdots \quad a_n:\kappa_n}{x[a_1,\ldots,a_n]:\gamma}\ \text{変項}$$

$$\frac{c@\gamma[\kappa_1,\ldots,\kappa_n] \quad a_1:\kappa_1 \quad \cdots \quad a_n:\kappa_n}{c[a_1,\ldots,a_n]:\gamma}\ \text{定項}$$

$$\frac{x_1@\kappa_1 \quad \cdots \quad x_n@\kappa_n \quad a:\gamma}{(x_1,\ldots,x_n)a:(\kappa_1,\ldots,\kappa_n)\gamma}\ \text{抽象体}$$

変項と定項の規則で $n=0$ のときには，$x[a_1,\ldots,a_n]$ ($c[a_1,\ldots,a_n]$) はそれぞれ x (c) のことであるとする．また抽象体の規則では $n>0$ でないといけない．アリティ o を持つ変項を対象変項といい，アリティ d を持つ変項を導出変項という．

簡単な例として定項 Human と Socrates のアリティをそれぞれ j[o]，o とすれば定項の規則により，

†1 通常はアリティ数のことをアリティと言うので，ここで言うアリティは通常のアリティの概念を拡張した概念となっている．

Human[Socrates] は判断のカテゴリー j に属する式になることがわかる．すなわち，不飽和なアリティ j[o] を持つ関数としての概念 Human を，それが期待する対象であるところのカテゴリー o をアリティとして持つ Socrates により補充することにより文 Human[Socrates] が判断のカテゴリー j に属する式として得られるのである．これに対して，Human や Socrates[Human] は上記規則により式として構成することができないため式から排除される．

ちなみに，基底カテゴリーからはじめて，関数型の構成子 $\rightarrow$ により構築される単純型 (σ で表す) を文法：

$$\sigma ::= \gamma \mid \sigma_1 \rightarrow \sigma_2$$

で定義すれば，カテゴリーと単純型は一対一に対応することがわかる．実際，Martin-Löf がかつて提案した式の理論 [11] はこのような単純型をもつ λ 計算と等価なものであった．しかし，このような型の定義のもとで式を構築しようとすると，どうしても

$$\frac{a : \sigma_1 \rightarrow \sigma_2 \quad b : \sigma_1}{a(b) : \sigma_2} \text{ 適用}$$

のような高階の式 a を分解する規則が式の構成規則として必要になる．なぜなら，これ以外に必要な以下の規則：

$$\frac{x@\sigma}{x : \sigma} \text{ 変項} \qquad \frac{c@\sigma}{c : \sigma} \text{ 定項}$$

$$\frac{x@\sigma_1 \quad a : \sigma_2}{(x)a : \sigma_1 \rightarrow \sigma_2} \text{ 抽象体}$$

だけでは，Human[Socrates] に対応する式を書くことができず，適用の規則を用いることにより Human(Socrates) と書くことができるからである．しかしながら，式をこのように定義すると，以下のように見掛けは異るが式としては同一視したくなるものができてくる．

$((x)x)(\mathtt{Socrates}) \equiv \mathtt{Socrates}$,

$(x)(\mathtt{Human}(x)) \equiv \mathtt{Human}$.

実際，Martin-Löf の式の理論やそれに影響を受けて提案された Edinburgh LF (Logical Framework) [7] では，λ 計算における β 規則と η 規則により書き換えることができる式を同一視することにしている．これに対して，我々の式の理論では，上の最初の例の左辺に相当する $((x)x)$[Socrates] や 2 番目の例の右辺に相当する Human は式として構成することができないが，最初の例の右辺や 2 番目の例の左辺に相当する式は書くことができる．

2.3 環境と具体化

単純型理論に基づく式の体系では β 規則の存在のため，式の操作においては代入操作 (substitution) が基本的である．我々の式の理論においては，代入操作は存在せずその代りに，環境 (environment) のもとで式を評価するという具体化 (instantiation) の操作が基本操作として必要になる．

x がアリティ κ を持つ変項で，a がカテゴリー κ に属する式であるとき，すなわち，$x@\kappa$ かつ $a : \kappa$ のとき，式 a は変項 x に割り当て可能であるという．また，このとき表現 $x = a$ を割り当て (assignment) と言う．$x_1, \ldots, x_k$ が相異なる変項のとき，割り当ての集合 $\rho = \{x_1 = a_1, \ldots, x_k = a_k\}$ を環境と言う．このとき ρ の定義域 (domain) $|\rho|$ を $\{x_1, \ldots, x_k\}$ と定める．

式 e と環境 ρ に対して，e の ρ による具体化 $[e]_\rho$ を以下のように定める．具体化の結果は式となるように定められる．

1. $[x]_\rho :\equiv e$ ($x@\gamma$ かつ $x = a \in \rho$ のとき).
2. $[x[a_1, \ldots, a_n]]_\rho :\equiv [a]_{\{x_1 = [a_1]_\rho, \ldots, x_n = [a_n]_\rho\}}$ ($n > 0$ かつ $x = (x_1, \ldots, x_n)a \in \rho$ のとき).
3. $[x[a_1, \ldots, a_n]]_\rho :\equiv x[[a_1]_\rho, \ldots, [a_n]_\rho]$ ($x \notin |\rho|$ のとき).
4. $[c[a_1, \ldots, a_n]]_\rho :\equiv c[[a_1]_\rho, \ldots, [a_n]_\rho]$.
5. $[(x_1, \ldots, x_n)[a]]_\rho :\equiv (x_1, \ldots, x_n)[[a]_\rho]$.

この定義がたしかに式を定めることは次のようにして確認することができる．カテゴリー κ の階数 (rank) $r(\kappa)$ を次のように定める．

1. $r(\gamma) = 0$.
2. $r((\kappa_1, \ldots, \kappa_n)\gamma) = \max\{r(\kappa_1), \ldots, r(\kappa_n)\} + 1$.

次に環境 ρ の階数を以下のように定める．

$$r(\rho) = \max\{r(\kappa) | x \in |\rho| \text{ and } x@\kappa\}.$$

ただし，空の環境の階数は 0 であるとする．このとき，$r(\rho)$ に関する帰納法を用いて，すべての式 e に

対して $[e]_\rho$ が定まることを示す．

$r(\rho) = 0$ のとき．このときは，上の定義で場合 2 は考えなくてよいので，式 e の構成に関する帰納法により $[e]_\rho$ の存在がわかる．

$r(\rho) > 0$ のとき．このときも式 e の構成に関する帰納法により $[e]_\rho$ の存在がわかる．ここで，問題になるのは場合 2 であるが，まず，構成に関する帰納法により

$$\rho' = \{x_1 = [a_1]_\rho, \ldots, x_n = [a_n]_\rho\}$$

が定まり，$r(\rho') < r(\rho)$ なので階数についての帰納法の仮定を用いて $[a]_{\rho'}$ が定まることがわかる．

具体化の操作がうまく定義でき停止するためには，変項がアリティを持つことが本質的である．アリティの制限なしに式を定義したとすれば，以下のように具体化の操作が停止しない例をつくることができる．

$$[x[x]]_{\{x=(y)[y[y]]\}} \equiv [y[y]]_{\{y=[x]_{\{x=(y)[y[y]]\}}\}}$$
$$\equiv [y[y]]_{\{y=(y)[y[y]]\}} \equiv \cdots$$

このようなことが実際には起きないのは，上の変項 x に整合的なアリティを付与できないからである．今 x にアリティを付与できたと仮定して，$x[x]$ を $x_1[x_2]$ と書くことにすれば，適当な基底カテゴリー γ とカテゴリー κ に対して，$x_1[x_2] : \gamma$， $x_1@\gamma[\kappa]$ かつ $x_2 : \kappa$ となる．したがって x_1 のアリティは $\gamma[\kappa]$，x_2 のアリティは κ となるが，これは矛盾である．

3 自然枠組

自然枠組は，前節で紹介した式を器として，判断と導出を可能な限り自然に記述することを目標として構築された形式的言語体系である．数学的な判断を形式的な言語で記述したものを，数理論理学の用語では「論理式」と呼ぶことが標準的であるが，自然枠組では「判断」という言葉をそのまま使うことにしている．その理由は，自然枠組は，我々が自然言語を用いて日常的に行っている判断活動 (すなわち，思考活動) をできるだけ自然に形式化することにより，人間の思考活動を支援する計算機環境を提供することを目指しているからである．

3.1 判断と導出

NF/CAL システムでは，数学の様々な理論や体系を導出ゲーム (derivation game) として実現する．各導出ゲームは，そこで用いる定項をそのアリティとともに宣言し，さらにそれらの定項を特徴づける公理を推論規則として与えることにより定められる．前節で準備した式は，多様なゲームでの判断および導出を一様な形式で簡潔に表現することを可能にする．以下では判断と導出の一般的な定義を与える．

各導出ゲームにおいて，カテゴリー j に属する式を判断と言う．たとえば，定項 $\Rightarrow$ @j[j, j] および $\forall$@j[(o)j] を持つ導出ゲームでは，H，J が判断のとき $\Rightarrow[H, J]$，$\forall[(x)J]$ は判断となる．$\Rightarrow[H, J]$ の形の判断を条件判断といい $H \Rightarrow J$ と表記する．$\forall[(x)J]$ の形の判断を普遍判断といい $\forall(x)J$ と表記する．条件判断および普遍判断は計算に関する導出ゲームの定義には不要な場合もあるが，多くのゲームで必要になる判断形式であるので NF/CAL システムではすべてのゲームにあらかじめ組み込んである．

NF においては，導出を一定の文脈のもとでなされる思考活動であるとして形式化する．この文脈は導出の作成過程において現在利用可能な仮定を与えるもので，導出文脈 (derivation context) と呼ばれる．導出文脈は次のように定義される．

定義 **1** $H_1, \ldots, H_n$ $(n \geq 0)$ が判断であり，$X_1, \ldots, X_n$ が相異なる導出変項であるとき，

$$X_1 :: H_1, \cdots, X_n :: H_n \vdash [\,]$$

の形の表現を導出文脈という．各 $X_i :: H_i$ をこの導出文脈の仮定 *(assumption)* という．Γ が上の形の導出文脈で J が判断であるとき，

$$X_1 :: H_1, \cdots, X_n :: H_n \vdash J$$

の形の表現を仮定付判断 *(hypothetical judgment)* といい，$\Gamma[J]$ または $\Gamma \vdash J$ と表記する．

次に規則の概念を定義する．$H_1, \ldots, H_n$ および J が判断で，$\langle x_1@\kappa_1, \ldots, x_m@\kappa_m \rangle$ がこれらの判断の自由変項を列挙したものであり，c がアリティ $\mathsf{d}[\kappa_1, \ldots, \kappa_m, \mathsf{d}, \ldots, \mathsf{d}]$ を持つ定項でそのアリティ数が $m + n$ であるとき以下の表現 R：

$$\langle\langle H_1, \ldots, H_n \rangle, J, c, \langle x_1, \ldots, x_m \rangle\rangle$$

のことを規則 (rule) という．規則 R は以下のように

表記することもある．

$$\frac{H_1 \quad \cdots \quad H_n}{J}\ c(x_1,\ldots,x_m).$$

この規則は H_1 から H_n までの判断をすべて導くことができたならば判断 J を導いてよいという内容の推論規則である．この規則はまたホーン節の表記法を借りて次のように簡潔に表記することもできる．

$$c(x_1,\ldots,x_m) :: J \text{ :- } H_1,\cdots,H_n.$$

上に定義した規則は厳密には規則のスキーマと称すべきもので，これから定義する導出においては，その具体例を用いて導出が作成される．すなわち，R が上の形の規則で $\rho = \{x_1 = e_1,\ldots,x_m = e_m\}$ が環境のとき，メタ表現：

$$\langle\langle [H_1]_\rho,\ldots,[H_n]_\rho\rangle,[J]_\rho,c,\langle e_1,\ldots,e_m\rangle\rangle.$$

を規則 R の環境 ρ の下での具体例 (instance) といい，$[R]_\rho$ と表記する．

各 R_i $(1 \le i \le n)$ が相異なる規則の名前を持つ規則であるとき，組 $\langle R_1,\ldots,R_n\rangle$ のことを導出ゲーム (derivation game) という．導出ゲーム G が与えられたとき，G のもとでの導出 (G 導出) を以下のように定義することができる．

定義 **2**導出ゲーム G，判断 J と導出文脈 Γ が与えられたときメタ判断「D は Γ のもとでの J の G 導出である」を以下のように定める．

1. 導出変項．X@d が導出変項で，$X :: H$ が Γ 中の仮定であるとき

   $$X$$

   は Γ のもとでの H の G 導出である．

2. 規則適用．R が G の規則，$D_1,\ldots,D_n$ が，それぞれ，Γ のもとでの $H_1,\ldots,H_n$ の G 導出であり，

   $$\frac{H_1 \quad \cdots \quad H_n}{J}\ c(e_1,\ldots,e_m)$$

   が R の具体例ならば

   $$c[e_1,\ldots,e_m,D_1,\ldots,D_n]$$

   は Γ のもとでの J の G 導出である．この導出を

   $$\frac{D_1 \quad \cdots \quad D_n}{J}\ c(e_1,\ldots,e_m).$$

   と表記することもある．

3. 条件導出．D が導出文脈 $\Gamma, X :: H$ のもとでの J の G 導出ならば

   $$\texttt{CD}[H,(X)D]$$

   は Γ のもとでの $H \Rightarrow J$ の G 導出である．この導出を $(X :: H)D$ と表記することもある．

4. 普遍導出．D が Γ のもとでの J の G 導出で x@o が Γ に自由に出現しない変項ならば

   $$\texttt{UD}[(x)D]$$

   は Γ のもとでの $\forall(x)J$ の G 導出である．導出について議論をしていることがあきらかな場合には，この導出を $(x)D$ と表記することもある．

上の定義で，CD および UD はそれぞれアリティ d[j,(d)d] および d[(o)d] を持つ定項である．このことから，導出はすべてカテゴリー d に属する式になることがわかる．

条件導出では J を導出した文脈と，それから新しく導かれた判断 $H \Rightarrow J$ を導出する文脈とが異ることに注意しておく．これは自然演繹の体系 [6] [12] で仮定を除去する規則 ($\Rightarrow$ の導入規則) に相当するものである．仮定が除去されたことを反映して得られた導出では X が抽象化されている．NF で推論のステップで導出文脈が変化するのはこの条件導出による推論だけである†2．普遍導出は自然演繹の体系における $\forall$ の導入規則に相当するものである．

次節で概要を紹介する CAL では D が導出文脈 Γ のもとでの J の G 導出であるというメタ判断を

$$\Gamma \vdash J \texttt{ in } G \texttt{ since } D$$

で表す．導出ゲームおよび導出の具体例についても次節で紹介する．

4 CAL システム

CAL システムは図 1 の第 1 階層 (最下位の階層) である Emacs Lisp を用いて，その上位にある第 2 階層から第 4 階層までを実装したシステムである．2 節では第 2 階層，3 節では第 3 階層について述べた．本節では第 4 階層が提供する導出ゲームについて述べる．第 4 階層はユーザが CAL システムを通してアクセスすることのできる階層である．Emacs Lisp の中で CAL システムを起動するコマンドを実行することに

†2 普遍導出においても変項の束縛が発生するが，ここで束縛される変項は導出文脈のさらに外のメタレベルでの文脈で暗黙のうちに宣言されていると考えることができる．

より，第 4 階層にアクセスするための専用のバッファが生成される．以後，ユーザはバッファの中で CAL のコマンドを実行することによりシステムとインタラクトすることができる．コマンドは導出ゲームを新しく定義したり，定義済みのゲームをプレイし，様々な判断の導出を作成したり，導出の正しさを検証するためのものが用意されている．

4.1 導出ゲーム

簡単な導出ゲームの例として，自然数を定義するゲーム Nat を以下のように定義してみよう．

$$\mathtt{Nat} :\equiv \langle\ \mathtt{zero} \quad :: \quad \mathtt{0 : Nat :-}\ ,$$
$$\mathtt{succ}(n) \quad :: \quad \mathtt{S}(n)\mathtt{ : Nat :-}\ n\mathtt{ : Nat}\ \rangle.$$

個々の導出ゲームは，ひとつの数学的体系や理論を定めるものであり，図 1 に示したように，そのゲームによりつくり出される「数学的世界」に存在する対象とそれらがどのような概念にあてはまる対象であるかを定めている．対象は対象のカテゴリー o に属する式により表現され，概念は概念のカテゴリー j[o] に属する式により表現される．ゲーム Nat が扱う対象は自然数 $(0, 1, 2, \dots)$ であり，概念は「□ は自然数である」という概念である．このことを反映して，ゲーム Nat の定項は zero@d，0@o，Nat@j[o]，succ@d[o, d]，n@o および S@o[o] と定めてあり，n : Nat，s(n) は，それぞれ Nat[n]†3，s[n] の省略形である†4．

CAL では導出ゲームを定義するコマンド defgame が用意されている．defgame は引数を 2 つとるコマンドであり，最初の引数は定義しようとするゲームの名前であり，今の場合は Nat である．2 つめの引数としてはゲームの定義本体を与える．定義本体は，(1) ゲームで用いる定項の宣言，(2) ゲームに固有の構文の定義，(3) ゲームに固有の規則の定義の 3 要素から構成される．

定項はアリティとともに宣言される．本稿では詳細を省略するが CAL では独自の文法記述言語を用いてユーザが自由にゲームに固有の構文を定義することができる．(1)，(2) で定めた構文を用いて，ゲームの規則は上に挙げた表現通りに定義することができる．

CAL の構文解析器は通常は与えられた文字列を解析し，成功した場合には NF の式を結果として返すが，解析のさいにモードを指定することにより様々な解析の結果を得ることを可能にしている．例えば，上の Nat の 2 つの規則はディフォールトのモードでは NF の式として解析されるがモードとして TEX モードを指定すれば，LATEX で処理可能なソーステキストを解析の結果とすることも可能である．以下はそのようにして得られたテキストを LATEX で処理して得られたものである．

$$\frac{}{\mathtt{0 : Nat}}\ \mathtt{zero} \qquad \frac{\boldsymbol{n}\mathtt{ : Nat}}{\mathtt{S(}\boldsymbol{n}\mathtt{) : Nat}}\ \mathtt{succ}$$

定義 2 にあるように，導出ゲームのもとでの導出は 4 種類のメタ規則により構築される．CAL では，これらを，導出変項，条件導出，普遍導出については，それぞれ，X，$(X :: H)[D]$，$(x)[D]$ の形式で表記し，規則適用については，

$$J \text{ by } c(e_1, \dots, e_m)\ \{D_1 ; \dots ; D_n\}$$

の形式で表記する．ただし，規則の名前の後に書くパラメータについては CAL の高階パターンマッチング機能により復元できるため省略してもよい．

以下は導出ゲーム Nat での正しい導出の例である．

```
⊢ ∀(x)[x:Nat ⇒ S(S(x)):Nat] in Nat since
(x)[
  (X::x:Nat)[
    S(S(x)):Nat by succ {
      S(x):Nat by succ {X} } ] ]
```

4.2 λ 計算

次に計算に関係する導出ゲームの例として，型のない λ 計算 LambdaBeta を定義してみよう．LambdaBeta は図 2 にある 7 つの規則からなるゲームとして定義できる． ここで，定項のアリティは，λ@o[o]，Λ@j[o]，red@j[o, o] であり，$\lambda x.\, M[x]$，$M : \Lambda$，M red P がそれぞれ $\lambda[(x)[M[x]]]$，$\Lambda[M]$，red$[M, P]$ の省略形となるように構文規則を定めることができる．$M : \Lambda$ は M が λ 項であるという判断を表し，M red P は

†3 「n は自然数である」と読む．

†4 したがって「:」は構文解析のための目印のためだけにあり，それを用いて述べられる判断の意味には寄与しない記号であることがわかる．

$$
\begin{aligned}
\langle\ \mathtt{abs}(M) &:: \lambda x.\, M[x] : \Lambda \ \mathtt{:-}\ \forall(x)[x : \Lambda \Rightarrow M[x] : \Lambda],\\
\mathtt{app}(M,N) &:: M\ N : \Lambda \ \mathtt{:-}\ M : \Lambda, N : \Lambda,\\
\mathtt{beta}(M,N) &:: (\lambda x.\, M[x])N\ \mathtt{red}\ M[N] \ \mathtt{:-}\ \lambda x.\, M[x] : \Lambda, N : \Lambda,\\
\mathtt{cong1}(M,N,P,Q) &:: M\ N\ \mathtt{red}\ P\ Q \ \mathtt{:-}\ M\ \mathtt{red}\ P, N\ \mathtt{red}\ Q,\\
\mathtt{cong2}(M,P) &:: \lambda x.\, M[x]\ \mathtt{red}\ \lambda x.\, P[x] \ \mathtt{:-}\ \forall(x)[x : \Lambda \Rightarrow M[x]\ \mathtt{red}\ N[x]],\\
\mathtt{refl}(M) &:: M\ \mathtt{red}\ M \ \mathtt{:-}\ M : \Lambda,\\
\mathtt{trans}(M,N,P) &:: M\ \mathtt{red}\ P \ \mathtt{:-}\ M\ \mathtt{red}\ N, N\ \mathtt{red}\ P\ \rangle
\end{aligned}
$$

図 2　導出ゲーム LambdaBeta の規則

λ 項 M が 0 回以上の書き換えにより λ 項 P に簡約されるという判断を表す．

最初の 2 つの規則が λ 項を生成する規則であり，以下のように表記することもできる．

$$\frac{\forall(x)[x : \Lambda \Rightarrow \boldsymbol{M}[x] : \Lambda]}{\lambda x\,.\,\boldsymbol{M}[x] : \Lambda}\ \mathtt{abs}$$

$$\frac{\boldsymbol{M} : \Lambda \quad \boldsymbol{N} : \Lambda}{\boldsymbol{MN} : \Lambda}\ \mathtt{app}$$

比較のため，通常の λ 項の定義を述べると以下のようになる．

1. 変項は λ 項である．
2. x が変項で M が λ 項ならば $\lambda x.\, M$ は λ 項である．
3. M，N が λ 項ならば $M\ N$ は λ 項である．

通常の定義では「変項は λ 項である」という規則が項を生成する出発点となる規則であり，この規則から出発して，より複雑な項を生成できるようになっている．LambdaBeta では上の 1 に相当する規則は存在しない．また規則 2 のパラメータは変項 x と λ 項 M のふたつであるが，この規則に対応する abs 規則のパラメータは高階のアリティo[o] を持つ M だけである．このような違いの結果，通常の定義では自由変項を持つ λ 項が生成されるのに対して，LambdaBeta では閉じた λ 項だけが生成されるようになっている．LambdaBeta で「自由変項」を持つ項を扱いたいときには，それらが λ 項であるという仮定を持つ導出文脈を設定することにより，その文脈の中での項として扱うことができる．すなわち，これらの「変項」は文脈の中でのみ自由な存在であり，「λ 項とは何か」という存在論的な問に対して LambdaBeta は閉じた λ 項を答として与えているのである．

残りの 5 つが λ 項の計算規則である．このうち実際に計算を進めるのは規則 beta であり，通常の定義での β 規則に対応する．ここでも M は高階のアリティo[o] を持つパラメータであり，代入ではなく具体化の操作により簡約が行われる．

以下は LambdaBeta における簡約の導出の例である．

```
⊢ λy.(λx.y)y red λy.y in LambdaBeta since
λy.(λx.y)y red λy.y by cong2 {
  (y)[(Y::y:Λ)[
      (λx.y)y red y by beta {
        λx.y : Λ by abs {(x)[(X::x:Λ)[Y]]};
        Y
      }]]}
```

4.3　述語論理

論理に関する導出ゲームの例として直観主義述語論理の体系 Pred を示す．Pred の定項とそのアリティは次のようである．規則の名前も定項であるが，これらの定項のアリティは復元可能なので省略する．

$$\bot\ @\mathtt{j}, \neg@\mathtt{j}[\mathtt{j}], \wedge@\mathtt{j}[\mathtt{j},\mathtt{j}], \vee@\mathtt{j}[\mathtt{j},\mathtt{j}], \exists@\mathtt{j}[(\mathtt{o})\mathtt{j}]$$

Pred は図 3 に示す 15 個の規則からなるゲームである． これらの規則で用いられるパラメータ P, Q, R はアリティj を持ち，a のアリティは o である．ただし，量化に関係する最後の 4 つの規則での P は高階のアリティj[o] を持つパラメータであるとする．規則 $\Rightarrow$-I および規則$\forall$-I は結論と前提の判断が一致する

$$
\begin{array}{rcl}
\langle\ \Rightarrow\texttt{-I}(P,Q) & :: & P \Rightarrow Q \texttt{ :- } P \Rightarrow Q, \\
\Rightarrow\texttt{-E}(P,Q) & :: & Q \texttt{ :- } P \Rightarrow Q, P, \\
\neg\texttt{-I}(P) & :: & \neg P \texttt{ :- } P \Rightarrow \bot, \\
\neg\texttt{-E}(P) & :: & \bot \texttt{ :- } \neg P, P, \\
\wedge\texttt{-I}(P,Q) & :: & P \wedge Q \texttt{ :- } P, Q, \\
\wedge\texttt{-EL}(P,Q) & :: & P \texttt{ :- } P \wedge Q, \\
\wedge\texttt{-ER}(P,Q) & :: & Q \texttt{ :- } P \wedge Q, \\
\vee\texttt{-IL}(P,Q) & :: & P \vee Q \texttt{ :- } P, \\
\vee\texttt{-IR}(P,Q) & :: & P \vee Q \texttt{ :- } Q, \\
\vee\texttt{-E}(P,Q,R) & :: & R \texttt{ :- } P \vee Q, P \Rightarrow R, Q \Rightarrow R, \\
\bot\texttt{-E}(P) & :: & P \texttt{ :- } \bot, \\
\forall\texttt{-I}(P) & :: & \forall(x)P[x] \texttt{ :- } \forall(x)P[x], \\
\forall\texttt{-E}(P,a) & :: & P[a] \texttt{ :- } \forall(x)P[x], \\
\exists\texttt{-I}(P,a) & :: & \exists(x)P[x] \texttt{ :- } P[a], \\
\exists\texttt{-E}(P,Q) & :: & Q \texttt{ :- } \exists(x)P[x], \forall(x)[P[x] \Rightarrow Q]\ \rangle
\end{array}
$$

図 3 導出ゲーム Pred の規則

規則であり，これらの規則を除いても導出可能な判断は変らないことがわかる[†5]．これらの規則で前提となる判断はすべての導出ゲームに共通の条件導出および普遍導出の規則を用いて導くことができ，これらが実質的な $\Rightarrow$ および $\forall$ の導入規則となっている．

Pred では排中律の二重否定を以下のように導出することができる．

```
⊢ ¬¬(A∨¬A) in Pred since
¬¬(A∨¬A) by ¬-I {
  (X::¬(A∨¬A))[
    ⊥ by ¬-E {
      ¬¬A by ¬-I {
        (Y::¬A)[
          ⊥ by ¬-E {
            X;
            A∨¬A by ∨-IR {Y}
          }]};
      ¬A by ¬-I {
        (Z::A)[
          ⊥ by ¬-E {
            X;
            A∨¬A by ∨-IL {Z}
          }]}}]}
```

この導出を CAL の構文解析器により TEX モードで解析することにより，図 4 のような通常の証明図に近い形式で表示することができる．この導出ではアリティj の変項 A が自由に出現していることがわかる．そのため，これはより正確には導出のスキーマとなっている．A の存在論的身分については次節で説明する．

5 おわりに

本稿で紹介してきた NF/CAL システムは教育用システムとして開発されてきたが，当初から数学の証明やプログラムの開発をコンピュータ上で支援するためのシステムとしても利用することを念頭に置いて開発してきた．

コンピュータ上での証明支援システムには様々なものがある (例えば，[10] [2] [7] [1] [3]) が，それらの多くは，1960 年代の後半に開発された Automath [9] の影響を受けている．これらのシステムの多くは型理論に基づいており，もっぱら専門家による使用を前提としている．そのためシステムの利用には型理論の知識が必要となり，学部で計算や論理を教育するためのシステムとしては不適当である．本稿で紹介した式の理論は，計算や論理の基礎的事項を，最初から説明するための必要性から考案したものであるが，同時に我々が日常的に用いる自然言語が本来持っている構造を自然に反映したものになっている．この考察はフレーゲ [5] による判断の分析の影響も受けており[†6]，NF における判断および導出の「形式的表現」は日常言語を連続的に拡張したものとみなすことができる．

上で述べた，日常言語の連続的な拡張[†7]とは次のよ

†5 なくてもすむ規則であるが，伝統的な自然演繹での導出と同型な導出を記述できるようにするために入れてある

†6 [16] にフレーゲの分析が式の理論の構築にどのような影響を与えたかの記述がある．

†7 フレーゲが概念記法 (Begriffsschrift [4]) を考えたのも日常言語の拡張として曖昧性ない厳密な言語を提供するためであったと考えることができる ([16] 参照)．

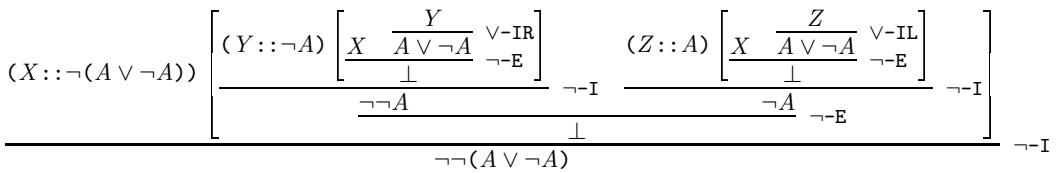

図 4　排中律の二重否定の導出

うなことである．数学の歴史を見るとき，数学の重要な発展は新しい言語表現を日常言語に追加することによりなされてきていることが観察される．自然数の 10 進表記や数の上の演算の表記等のそれまで日常言語になかった表現が発明，追加され国際的にも共通の表記により数学が語られている．たとえば，「$a = b$ ならば $b = a$」は「a と b が等しいならば b と a は等しい」と読むことができる日本語の文となる．自然枠組は，この歴史的観察をもとにして，日常言語の延長として判断を形式的に表現する枠組を実現することを目指して構築されたものである．

「連続性」の帰結として，4.3 節での導出 (図 4) における自由変項 A はその値として日常言語における数学的判断をとることのできる変項であり，図 4 中での使用に先立って A を判断のアリティj を持つ変項として暗黙に宣言していることがわかる．すなわち，この導出は「任意の判断 A に対して $\neg\neg(A \vee \neg A)$ である」という判断の正しさの根拠を与えているのである．この例についてもあてはまることであるが，NF/CAL システムでは「変項は (1) 宣言をしてから，その (2) 有効範囲 (scope) で (3) 使用する」という，基本的であるが十分に強調されていない原則を重視しており，この原則を変項 **3** 原則と呼んでいる．仮定を除去する規則についても，通常の自然演繹の体系では仮定に番号を付けて除去される仮定を参照しているが，NF/CAL では仮定は明示的に変項を用いて宣言し，$[\cdots]$ で示される有効範囲の中で用いた仮定は，有効範囲を括弧 (]) で閉じることにより明示的に除去される．

現代的な数理論理学の講義においては，論理体系，たとえば述語論理の体系は形式言語により記述される「外国語」として教授される．そこでは，外国語教育と同じく，述語論理のための形式言語を対象言語，述語論理についての説明をするための日常言語 (たとえば日本語) をメタ言語として教育がなされる．その結果，対象言語を用いて語られる数学とメタ言語を用いて語られる数学の二重構造が発生し，図 4 の導出はメタ言語においては「任意の論理式 A に対して $\neg\neg(A \vee \neg A)$ は述語論理における定理である」というメタ定理として語られることになる．これに対して，すでに見てきたように NF/CAL を用いる「計算と論理」の講義では述語論理は形式化された「国語」(母国語) で記述されたゲーム Pred として教授される．ここでは，国語教育と同じく，同一の言語 (我々の場合は日本語) がメタ言語と対象言語として用いられるのである†8．

NF 上のゲーム Pred が (拡張された) 日本語であるということにより Pred は開かれた体系 (open system) であることも帰結する．すなわち，Pred で導かれた定理「任意の判断 A に対して $\neg\neg(A \vee \neg A)$ である」において A として日本語で書かれた (数学的) 判断を自由にとることができ，この判断は体系 Pred の外にある判断である．これに対して，通常の数理論理学における述語論理の体系での対応するメタ定理においては，A としてとることができるのは述語論理の体系の中にある論理式に限られる†9．

以上見てきたように，NF/CAL システムは，型理論に基く既存の証明支援システムが型の概念を基礎において構築されているの対して，より基礎的な概念である判断と導出の概念およびこれらの基礎的概念を適切に表現するための式の理論をもとに階層的に構築さ

†8　小学校で「1+1」を「いちたすいち」と読むように教えるのは算数の教育と同時に国語教育もしているのである．

†9　外国語が母国語に翻訳可能なように，論理式 A を意味論的に解釈することにより日常言語で書かれた判断に適用することは可能である．

れており，計算と論理に関する人間の思考活動を支援するための自然な枠組を提供するものになっている．

最後に，本稿で示した例は現在実装されている NF/CAL システムで動かした結果をそのまま取り込んだものであるが，そのベースとなっている式の理論は本稿で紹介した式の理論よりは一世代古いものであることを断わっておきたい[†10]．現在，新しい式の理論をベースにしたシステムの実装が進行中であり，またこれと並行して最下位の階層を Haskell で置き換えた実装も進行中である．これらの実装が完了すれば NF/CAL システムを，できるだけ早くフリーソフトウェアとして一般に公開し，教育用システムないし証明支援システムとして多くの方々に使用してもらえればと考えている．

参 考 文 献

[1] Benl, H., Berger, U., Seisenberger, M., Schwichtenberg, H. and Zuber, W.: Proof theory at work: Program development in the Minlog sytem, in *Automated Deduction*, Bibel, W. and Schmitt, P.H. eds., Vol II, Kluwer, 1998.

[2] Bertot, Y. and Castéran, P.: *Interactive Theorem Proving and Program Development, Coq'Art: The Calculus of Inductive Constructions*, Texts in Theoretical Computer Science, Springer, 2004.

[3] Buchberger, B., Dupre, C., Jebelean, T., Kriftner, F., Nakagawa, K., Vasaru, D. and Windsteiger, W.: The Theorema Project: A Progress Report, in *Symbolic Computation and Automated Reasoning (Proceedings of CALCULEMUS 2000, Symposium on the Integration of Symbolic Computation and Mechanized Reasoning, August 6-7, 2000, St. Andrews, Scotland)*, Kerber, M. and Kohlhase, M. (eds.), A.K. Peters, Natick, Massachusetts, pp. 98-113.

[4] Frege, G.: *Begriffsschrift, eine der arithmetischen nachgebildete Fromelsprache des reinen Denkens*, Halle a.S., 1879; 藤村龍雄 (編) フレーゲ著作集 1『概念記法』，勁草書房，1999.

[5] Frege, G.: *Grundgesetze der Arithmetic*, 1893; 野本和幸 (編) フレーゲ著作集 3『算術の基本法則』，勁草書房，2000.

[6] Gentzen, G.: Untersuchngen über logishce Shließen, *Mathematishe Zeitschrift*, Vol. 39 (1935), pp. 176–210; Investigations in logical deductions, in Szabo, M.E. (ed.), *The Collected Papers of Gerhard Gentzen*, North-Holland (1969), pp. 68–131.

[7] Harper, R., Honsell, F. and Plotkin, G., A Framework for Defining Logics, *J.ACM*, Vol. 40 (1993), pp. 143–184.

[8] McCarthy, J.: Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I, *Comm. ACM*, Vol. 3 (1960), pp. 184–195.

[9] Nederpelt, R.P., Geuvers, J.H. and de Vriijer, R.C. (eds.): *Selected Papers on Automath*, Sudies in Logic and the Foundation of Mathematics, Vol. 133, North-Holland, 1994.

[10] Nipkow, T., Paulson, L.C. and Wenzel, M.: *Isabelle/HoL, A Proof Assistant for Higher-Order Logic*, LNCS 2283, Springer, 2002.

[11] Nordström, B., Petersson, K. and Smith, J.M., *Programming in Martin-Lö's Type Theory, An Introduction*, Clarendon Press, Oxford, 1990.

[12] Prawitz, D.: *Natural Deduction, A Proof-Theoretic Study*, Dover, 2006.

[13] Sato, M., Kameyama, Y. and Takeuti, I.: CAL: A computer assisted learining system for computation and logic, in Moreno-Diaz, R., Buchberger, B. and Freire, J-L. (eds.), *Computer Aided Systems Theory – EUROCAST 2001*, LNCS 2718, pp. 509–524, Springer, 2001.

[14] Sato, M.: Theory of Judgments and Derivations, in Arikawa, S. and Shinohara, A. (eds.), *Progress in Deiscovery Science*, LNCS 2281, pp. 78–122, Springer, 2002.

[15] Sato, M.: A Simple Theory of Expressions, Judgments and Derivations, in Maher, M.J. (ed.), *Advances in Comupter Science – ASIAN 2004*, LNCS 3321, pp. 437 – 451, Springer, 2004.

[16] 佐藤雅彦: フレーゲの計算機科学への影響，科学哲学，Vol. 38, No. 2 (2005), pp. 21–33.

†10 NF/CAL の変遷については文献 [14] [13] [15] を参照．