知識工学と人工知能 52－7
（1987. 5. 7）

Lazy Evaluationをもつ非決定性LISPの意味論

島田 陽一, 山崎 進, 堂下 修司
京都大学工学部情報工学教室

Lazy evaluationをもつ非決定性LISP$\mathcal{L}$を定義し，$\mathcal{L}$におけるプログラムの操作的意味およびそれに対応する表示的意味の構成方法の概要を示す．操作的意味としては，プログラムの停止性および停止した場合の計算結果に注目する．また，$\mathcal{L}$の表示的意味を定義する際に基礎となる領域においては，非停止を表す元を無限個用意することによって，部分式に対するプログラムの実行が停止しない場合についても，式に正しい解釈を与えるようにする．そして，この2つの意味が等価であることの証明に特殊な帰納法が適用されることに関して，その概要を説明する．

SEMANTICS OF A NONDETERMINISTIC LISP WITH LAZY EVALUATION

Youichi SHIMADA, Susumu YAMASAKI and Shuji DOSHITA
Department of Information Science, Faculty of Engineering, Kyoto University
Yoshida-hommachi, Sakyo, Kyoto, 606, Japan

We define a nondeterministic LISP $\mathcal{L}$ with lazy evaluation, and show the operational semantics of programs in $\mathcal{L}$ and the approximation of the method to construct the denotational semantics corresponding with it. As the operational semantics, we note the termination and the results of the execution of every program. In the domain for the denotational semantics, we have infinite elements to denote that the execution does not terminate, and give the correct interpretation to every expression even if the execution for a part of it does not terminate. And we explain the approximation of the application of the special induction to prove the equivalence of these semantics.

1. はじめに

人工知能の分野では非決定的な探索が直接的に記述できる論理型言語が注目されている．この論理型言語のプログラムは，各述語のもつ引数の入出力的な役割が明確な場合，述語を関数とみなすことによって，非決定性をもつ関数型言語のプログラムに容易に変換できることが多い．また一般に，論理型言語においてはデータ構造としてリスト形式が頻繁に用いられるが，これは非決定性関数型言語においてもリスト形式に対応する．本稿ではlazy evaluationをもつ非決定性LISPを定式化し，その操作的意味に対応する表示的意味を構成する方法の概要を示す．[1]ではこれに関する形式的な記述を詳細に与えている．ただし，論理型言語との対応については特に考慮していない．

2. Lazy evaluation をもつ非決定性LISP$\mathcal{L}$

まず，本稿で対象とする非決定性LISP$\mathcal{L}$について簡単に説明する．

$\mathcal{L}$のもつ基本関数は car，cdr，cons，atom，equal，if，or，block の８個であり，それぞれの引数の個数は 1，1，2，1，2，3，2，0 である．一般的な純LISPと異なるおもな点は次のとおりである．

1. データの等価性を判定する基本関数として，eqの代わりにequalをもつ．

これは，データを物理的な構造ではなく，S式として扱うためである．

2. 条件式を表す基本関数として，condの代わりにifをもつ．

式if[x,y,z]は，'if x then y else z' を意味する．

3. 非決定的な選択を指示する基本関数orをもつ．

式or[x,y]に対してはx，yのいずれかが選択され実行される．

4. 実行を閉塞させる基本関数blockをもつ．

式blockが評価されると，実行が打ち切られ，値は生成されない．

実行の閉塞は非決定的な探索の失敗に対応するもので，一般にエラーとなる場合（たとえば，アトムAに対する式car[A]の評価）にも実行が閉塞される．

さらに，$\mathcal{L}$は次のような特徴をもつ．

5. Lazy evaluation をもつ．

たとえば，式car[cons[A,f[B]]]ではf[B]が評価されずに値Aが返される．

6. 引数の評価は並行外側規則[4]に従う．

たとえば，式cons[f[g[A]],h[B]]では，まずfとhが並行して評価される．

また，$\mathcal{L}$はλ式による表現をもたないので，$\mathcal{L}$のプログラムは

f[x]=or[NIL,g[x,f[x]]]

g[x,y]=cons[x,f[x]]

のように等式によって表される．

3. $\mathcal{L}$におけるプログラムの操作的意味

本稿では，$\mathcal{L}$におけるプログラムの操作的意味として，プログラムの停止性と停止した場合の計算結果を考える．ただし，$\mathcal{L}$のプログラムの停止とは，非決定的な選択がどのように行われても必ず停止するということを表す．

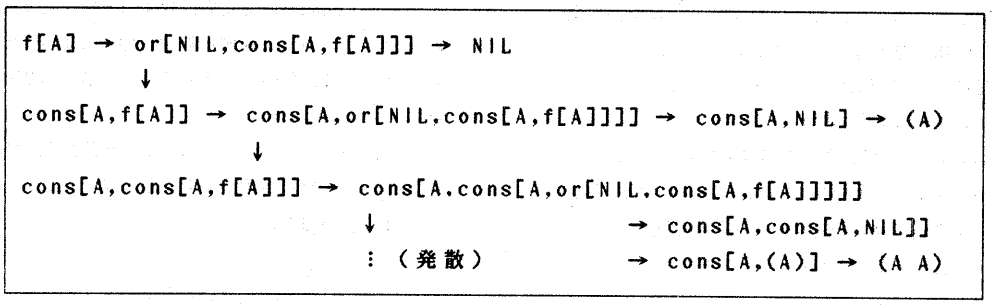

図１　f[A]に対する実行過程

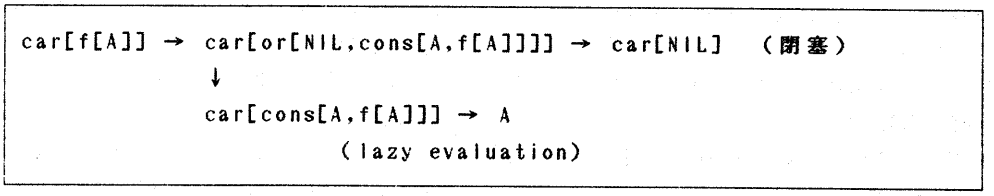

図２　car[f[A]]に対する実行過程

【例１】　プログラム

f[x]=or[NIL,cons[x,f[x]]]

において，式f[A]に対する実行は

停止性：なし

計算結果：NIL，(A)，(A A)，(A A A)，…

であり（図１），式car[f[A]]に対する実行は

停止性：あり

計算結果：A

となる（図２）．　□

４．　$\mathcal{L}$におけるプログラムの表示的意味

例１のプログラムに対して表示的意味がどのようになるかを考察する．このプログラムに対してxにAを代入すると次のようになる．

f[A]=or[NIL,cons[A,f[A]]]

ここで，式および基本関数の解釈を表す関数をそれぞれσ，μとすると

$$\sigma(f[A])$$
$$=\sigma(or[NIL,cons[A,f[A]]])$$
$$=\sigma(NIL)\cup\sigma(cons[A,f[A]])$$
$$=\{NIL\}\cup\{\mu(cons;x,y)\mid x\in\sigma(A),\ y\in\sigma(f[A])\}$$
$$=\{NIL\}\cup\{\mu(cons;A,y)\mid y\in\sigma(f[A])\}$$

が成立する（σの定義は省略）．すなわち，$\sigma(f[A])$は方程式

$$X=\{NIL\}\cup\{\mu(cons;A,y)\mid y\in X\}\quad (a)$$

の解である．

まず，値（アトムとリスト）全体の集合をDとして，(a)に対する$\mathcal{P}(D)$における解を考える．Dの任意の元yに対して

$\mu(cons;A,y)=(A.y)$

とすると，(a)の解は

$$X=\{NIL,\ (A),\ (A\ A),\ (A\ A\ A),\ \cdots\}\quad (b)$$

となる．これは，f[A]に対する実行の計算結果に

対応している．しかし，プログラムの停止性に関する情報は全く与えていない．

そこで，非停止を表す元⊥を導入し，

$D^* = D \cup \{\perp\}$

とおいて$\mathcal{P}(D^*)$における解を考える．ここで，f[A]に対する実行が停止しなければcons[A.f[A]]に対する実行も停止しないことに注意して，

$\mu(\mathrm{cons};A.\perp) = \perp$

とすると，(b)に加えて

$X = \{\perp, \mathrm{NIL}, (A), (A\ A), (A\ A\ A), \cdots\}$ (c)

も解となる．(b)は⊥を含まないのでf[A]に対する実行が停止することを表し，一方，(c)は停止しないことを表す．

実際には，f[A]に対する実行は停止しないので，(c)が妥当な解である．

ところが，解(c)を用いてcar[f[A]]に対する解釈を求めると次のようになる．

$\sigma(\mathrm{car}[f[A]])$
$= \{\mu(\mathrm{car};x) \mid x \in \sigma(f[A])\}$
$= \{\perp, A\}$

ただし，

$\mu(\mathrm{car};\perp) = \perp$，

$\mu(\mathrm{car};\mathrm{NIL}) =$ なし　(car[NIL]：閉塞)，

$\mu(\mathrm{car};(A)) = \mu(\mathrm{car};(A\ A)) = \cdots = A$

である．実際には，car[f[A]]に対する実行は停止するので，σ(car[f[A]])が⊥を含むことは妥当ではない．これは，lazy evaluation の効果がこの数学的モデルに全く反映されていないためである．すなわち，cons[A,f[A]]に対する実行が停止しなくても，car[cons[A.f[A]]] に対する実行は停止してAを出力するので，解(c)における⊥に対しては

$\mu(\mathrm{car};\perp) = A$

となるべきなのである．

この点を考慮して，D^*を拡張し，非停止を表す元を

$\perp, (\perp.\perp), (A.\perp), (\perp.A), (A\ A.\perp), \cdots$

のように可算無限個用意する．そして，μ(cons;A.⊥)の値を⊥ではなく(A.⊥)とすることによって，consの第２引数に対する実行は停止しないが，第１引数に対する実行は停止してAを出力するということを明示しておく．このようにすると

$\sigma(f[A]) = \{\mathrm{NIL}\} \cup \{(A.y) \mid y \in \sigma(f[A])\}$

より，f[A]に対する実行が停止しなくても，car[f[A]]の解釈は

$\sigma(\mathrm{car}[f[A]])$
$= \{\mu(\mathrm{car};x) \mid x \in \sigma(f[A])\}$
$= \{A\}$

となり，停止することを示すようになる．ただし，f[A]に対する実行が閉塞される場合にσ(f[A])が空集合にならないようにするために，閉塞を表す元*B*を用意して

$\sigma(f[A]) = \{*B*\}$

とする．

これで lazy evaluation の効果が式の解釈に反映されるようになったが，改めて方程式(a)の解を求めてみると，(b)は解であるが(c)は解ではないことがわかる．すなわち，妥当な解が失われてしまったことになる．この問題を解決するためには，解を求める空間をさらに拡張し，また同時に，妥当でない解を排除する必要がある．以下では，そのための方法を形式的に記述する．

【定義】　集合Tを以下のように定義する．

(1)　DはTの部分集合である．

(2)　⊥および*B*はTの元である．

(3)　xおよびyがTの元ならば，(x.y)はTの元である．

(4)　Tは以上の条件を満たす最小の集合である．

□

【定義】　T上の半順序関係$\sqsubseteq$を以下のように定義する．

(1)　Tの任意の元xに対して，$x \sqsubseteq x$ および $\perp \sqsubseteq x$ である．

(2)　Tの任意の元x_1，x_2，y_1，y_2に対して，$x_1 \sqsubseteq x_2$ かつ $y_1 \sqsubseteq y_2$ ならば，$(x_1.y_1) \sqsubseteq (x_2.y_2)$ である．

(3)　$\sqsubseteq$は以上の条件を満たす最小の関係である．

□

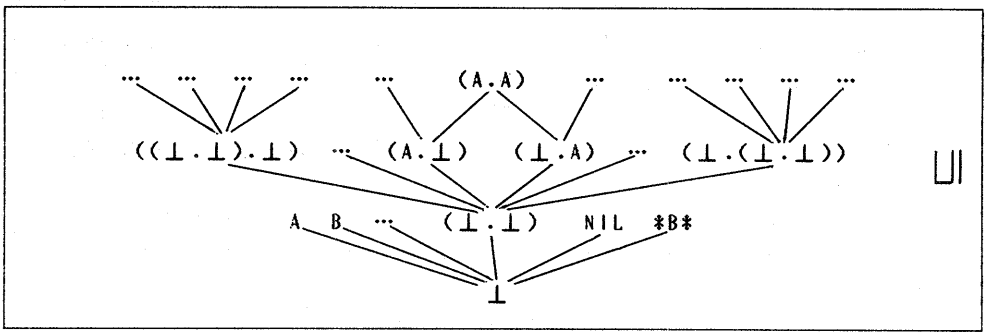

図３　Ｔの構造

集合Ｔの構造は図３のようになる．

【定義】　Ｔ上のorを除く基本関数の解釈μ_Tを以下のように定義する．

(1) $\mu_T(car;x)=\begin{cases} y & x=(y.z) \\ *B* & x=アトム，NIL，*B* \\ \bot & x=\bot \end{cases}$

(2) $\mu_T(cdr;x)=\begin{cases} z & x=(y.z) \\ *B* & x=アトム，NIL，*B* \\ \bot & x=\bot \end{cases}$

(3) $\mu_T(cons;x,y)=(x.y)$

(4) $\mu_T(atom;x)=\begin{cases} *T* & x=アトム \\ *F* & x=NIL，(y.z) \\ *B* & x=*B* \\ \bot & x=\bot \end{cases}$

(5) $\mu_T(equal;x,y)=\begin{cases} *T* & x=y：x,y\in D \\ *F* & x\neq y：x,y\in D \\ *B* & x,yのいずれか*B*を含む \\ \bot & その他 \end{cases}$

(6) $\mu_T(if;x,y,z)=\begin{cases} y & x=*T* \\ z & x=*F* \\ \bot & x=\bot \\ *B* & その他 \end{cases}$

(7) $\mu_T(block)=*B*$　□

基本関数orは非決定性をもつので，$\mu_T(or;x,y)$は定義されない．

【定義】　Ｔの空でない有限部分集合全体をＳとする．□

Ｔのべき集合の部分集合Ｓをこのように与えるのは，プログラムの実行において，非決定的な選択の可能性が有限時間では有限通りしか存在しないからである．

【定義】　Ｓ上の擬順序関係$\sqsubseteq_M$を以下のように定義する．

Ｓの任意の元X，Yに対して，

$$X\sqsubseteq_M Y \Leftrightarrow (\forall x\in X)(\exists y\in Y)\ x\sqsubseteq y \wedge (\forall y\in Y)(\exists x\in X)\ x\sqsubseteq y$$　□

ここで，$\sqsubseteq_M$はMilnerの関係[2]と呼ばれる関係であり，たとえば，

$$\{\bot\}\sqsubseteq_M\{\bot, A\}\sqsubseteq_M\{A\}$$

である．また，

$X=\{\bot, (A.A)\}$，

$Y=\{\bot, (\bot.\bot), (A.A)\}$

に対して，$X\sqsubseteq_M Y$ かつ $Y\sqsubseteq_M X$ であるので，$\sqsubseteq_M$は半順序関係ではない（図４）．

この関係$\sqsubseteq_M$は，実行が進むにつれてそれに対応する点列が単調増加になるという性質をもつ[2][3]（詳細は省略）．

【定義】　Ｓ上の基本関数の解釈μ_Sを以下のように定義する．

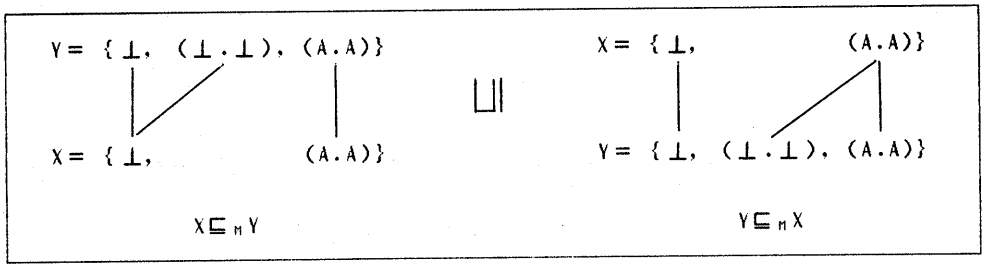

図４　Milnerの関係

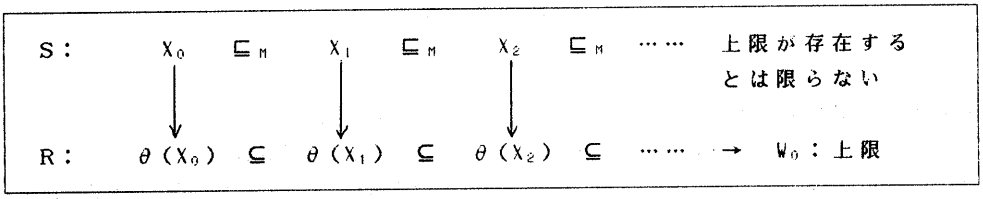

図５　ＳとＲの対応

(1)　$\mu_S(f;X_1,\cdots,X_n)$

$=\{\mu_T(f;x_1,\cdots,x_n)\mid x_1\in X_1,\ \cdots,\ x_n\in X_n\}$

（fはorを除く基本関数，nはfの引数の個数）

(2)　$\mu_S(or;X_1,X_2)=X_1\cup X_2$　□

この集合Ｓは有限時間内の実行に対応していて，無限時間後の実行結果はＳの元からなる単調増加点列 $X_0\sqsubseteq_M X_1\sqsubseteq_M X_2\sqsubseteq_M\cdots$ の上限に対応する．しかし，$\sqsubseteq_M$は半順序関係ではなく，一般にはＳに上限が存在するとは限らない．従って，Ｓの任意の元Ｘにイデアル

$\theta(X)=\{Y\in S\mid Y\sqsubseteq_M X\}$

を対応させることによってＳの完備化を行う．

【定義】　Ｓの$\sqsubseteq_M$に関するイデアル全体をＲとする．□

Ｒは集合の包含関係$\subseteq$に関して完備な半順序集合であり，Ｓの任意の元Ｘ，Ｙに対して

$X\sqsubseteq_M Y \Leftrightarrow \theta(X)\subseteq\theta(Y)$

であるので，無限時間後の実行結果は，上述の単調増加点列に対応する単調増加点列

$\theta(X_0)\subseteq\theta(X_1)\subseteq\theta(X_2)\subseteq\cdots$

の上限に対応させることができる（図５）．

【定義】　Ｒ上の基本関数の解釈μ_Rを以下のように定義する．

$\mu_R(f;W_1,\cdots,W_n)$

$=\{X\in S\mid(\exists Y_1\in W_1)\cdots(\exists Y_n\in W_n)\ X\sqsubseteq_M\mu_S(f;Y_1,\cdots,Y_n)\}$

（fは基本関数，nはfの引数の個数）　□

この関数μ_Rは連続であり，かつ

$\mu_R(f;\theta(X_1),\cdots,\theta(X_n))$

$=\theta(\mu_S(f;X_1,\cdots,X_n))$

が成立するように定義されている．

さて，改めて例１のプログラムにおける式f[A]の解釈を求めてみると次のようになる．

$\sigma(f[A])$

$=\sigma(or[NIL,cons[A,f[A]]])$

$=\mu_R(or;\sigma(NIL),\sigma(cons[A,f[A]]))$

$=\mu_R(or;\sigma(NIL),\mu_R(cons;\sigma(A),\sigma(f[A])))$

ここで，

$\sigma(A)=\theta(\{A\})=\{Y\in S \mid Y\sqsubseteq_M \{A\}\}$

$=\{\{\perp\},\{\perp,A\},\{A\}\}$

および

$\mu_S(cons;Y_1,Y_2)$

$=\{\mu_T(cons;x_1,x_2) \mid x_1\in Y_1,\ x_2\in Y_2\}$

$=\{(x_1.x_2) \mid x_1\in Y_1,\ x_2\in Y_2\}$

より

$\mu_R(cons;\sigma(A),\sigma(f[A]))$

$=\{X\in S \mid (\exists Y_1\in\sigma(A))(\exists Y_2\in\sigma(f[A]))$

$X\sqsubseteq_M \mu_S(cons;Y_1,Y_2)\}$

$=\{X\in S \mid (\exists Y_2\in\sigma(f[A]))$

$X\sqsubseteq_M \{(\perp.x_2) \mid x_2\in Y_2\}\}$

$\cup\{X\in S \mid (\exists Y_2\in\sigma(f[A]))$

$X\sqsubseteq_M \{(\perp.x_2),(A.x_2) \mid x_2\in Y_2\}$

$\cup\{X\in S \mid (\exists Y_2\in\sigma(f[A]))$

$X\sqsubseteq_M \{(A.x_2) \mid x_2\in Y_2\}\}$

$=\{X\in S \mid (\exists Y_2\in\sigma(f[A]))$

$X\sqsubseteq_M \{(A.x_2) \mid x_2\in Y_2\}\}$

ゆえに

$\sigma(f[A])$

$=\{X\in S \mid (\exists Y_1\in\sigma(NIL))$

$(\exists Y_2\in\mu_R(cons;\sigma(A),\sigma(f[A])))$

$X\sqsubseteq_M \mu_S(or;Y_1,Y_2)\}$

$=\{X\in S \mid (\exists Y_2\in\mu_R(cons;\sigma(A),\sigma(f[A])))$

$X\sqsubseteq_M \{NIL\}\cup Y_2\}$

$=\{X\in S \mid (\exists Y\in\sigma(f[A]))$

$X\sqsubseteq_M \{NIL\}\cup\{(A.x) \mid x\in Y\}\}$

となる．この方程式の最小解は次の連続変換

$\tau : R\to R$ の最小不動点に一致する．

$\tau(W)$

$=\{X\in S \mid (\exists Y\in W)$

$X\sqsubseteq_M \{NIL\}\cup\{(A.x) \mid x\in Y\}\}$

そして，Kleeneの第一帰納定理[4]により，τの最小不動点W_0は

$W_0=\cup\{\tau^k(\{\{\perp\}\}) \mid k\in\omega\}$

（$\{\{\perp\}\}$はRの最小元）

で与えられ，上述の解(c)は

$(\cup W_0\cap D)\cup\{\perp \mid (\forall X\in W_0)(\exists x\in X)$

xは⊥を含み*Bを含まない}

によって導くことができる．$\mathcal{L}$におけるプログラムの表示的意味はこのようにして与えられる．

（注意）　ここでは説明を簡単にするために式の解釈σを式のみを引数としてもつ関数として扱っているが，正確にはσはそれだけでなくプログラムで定義される関数の解釈および変数に対する付値を引数としてもつ関数である．また，連続変換τはR上の変換ではなく，プログラムで定義される関数の解釈全体の集合上での変換である．

5．　$\mathcal{L}$におけるプログラムの操作的意味と表示的意味の等価性の証明

前章では，$\mathcal{L}$におけるプログラムの表示的意味をその操作的意味に対応するように構成する方法の概要を示した．この議論を形式的に記述することによって，これらの２つの意味が正確に対応すること（等価性）を証明することが可能である[1]．本章では，この証明の特徴的な部分についてその概形を簡単に説明する．

ここで証明しようとする定理は，「$\mathcal{L}$の任意の式（正確には変数を含まない式）に対して，プログラムを実行した際の停止性および計算結果が，最小不動点意味におけるその式の解釈と対応している」というものである．ところで，$\mathcal{L}$の式はDの元と関数（基本関数とプログラムで定義される関数）から再帰的に構成されているので，一般に「任意の式に対して成立する」という内容の定理を証明する際には，式の構造に関する帰納法が用いられる．そして，プログラムの実行がステップ的に進行することと最小不動点が単調増加点列の上限として与えられることを考慮すると，結局，この証明には式の構造と自然数の構造に関する帰納法が適用されると予想される．

ところが，$\mathcal{L}$が lazy evaluation をもっているので，式eに対する実行が停止しなくても，式car[e]に対する実行は停止することがある．このため，式の構造に関する単純な帰納法では証明できない（図６）．さらに，プログラムの実行では

(a) 帰納段階が適用可能な例
[仮定] 式eに対する実行が停止してxを出力する.
[結論] 式car[e]に対する実行が停止してμ_T(car;x)を出力する.
(ただし, μ_T(car;x)=*B* ならば何も出力されない.)
(b) 帰納段階が適用不可能な例
[仮定] 式eに対する実行が停止しない.
[結論] 式car[e]に対する実行が停止してyを出力する.

図6 式の構造に関する帰納法における帰納段階の適用可能性

式の中に変数が現れない（図1，図2参照）のに対し，式の解釈の定義[1]（本稿では省略）では式の中に変数が現れるという点で，操作的意味と表示的意味をそれぞれ構成する際の式の構造に差異があり，結果的には，式の構造と自然数の構造に新たな2つの抽象的構造を加えて合計4つの構造に関する帰納法が適用されることになる．その上，プログラムで定義される関数に不動点解釈が与えられることを意図して式の解釈が定義されているにもかかわらず，証明では一般には不動点ではない単調増加点列の各点とプログラムの実行の各時点との対応を扱うことになるので，自然数の構造に関する単純な帰納法を適用することが不可能となる．従って，最終的には，この4つの構造をもつ空間に対して特別な半順序構造を導入することによって，帰納法による証明が可能となる．

6. 主要な結果

(1) 以上の議論を形式的に記述し，$\mathcal{L}$におけるプログラムの操作的意味と表示的意味が正確に対応することの証明を与えた[1]．これは[3]の結果の1つの拡張である．

(2) (1)の結果に基づいて表示的意味論に対応した非決定性LISP$\mathcal{L}$のインタプリタを与えることができる[1]．

参考文献

[1] Y.Shimada : Semantics of a Nondeterministic LISP with Lazy Evaluation, Master Thesis, Division of Information Science, Master Course of Graduate School of Engineering, Kyoto University (1987).

[2] G.D.Plotkin : A Powerdomain Construction, SIAM J. COMPUT. 5-3 (1976), 452-487.

[3] M.Hennessy and E.A.Ashcroft : The Semantics of Nondeterminism, In "Automata, Languages and Programming", 3rd International Colloquium, Edinburgh (1976), 478-493.

[4] Zohar Manna : Mathematical Theory of Computation (McGraw-Hill Inc. 1974), Chap.5.