ソフトウェア基礎論　27－5
プログラミング言語　19－5
（1988. 12. 9）

リフレクティブなPrologの形式化と意味論
A Formalization of Reflective Prolog and Its Semantics

菅野 博靖
Hiroyasu Sugano
富士通(株)国際情報社会科学研究所
International Institute for Advanced Study
of Social Information Science, FUJITSU LIMITED

あらまし：本稿では逐次論理型言語であるPrologにリフレクションを導入することを試みる。そのために、まずPrologの意味論を代入ストリームに基づく最小不動点として形式化する。この意味論は節の間の順序関係に基づいてPrologの逐次性を表現することができ、宣言的意味論と手続き的意味論の中間に位置するものだと言える。さらに、その上にリフレクションを導入することによってR-Prologを提案し、リフレクションの理論的枠組を提供することを試みる。

Abstract: Reflection is one of the most promising architecture for programming languages which are expected to be semantically transparent and powerful in their descriptive ability. In this paper we propose reflective Prolog (R-Prolog) and try to make a theoretical investigation on it. On that purpose, we also formalise a new fixed-point semantics of Prolog that is based on the concept of substitution stream. This semantics of Prolog is an intermediate one between the declarative and operational semantics of Prolog.

1　はじめに

プログラムの挙動に関するさまざまな性質について論じるためには、それが書かれているプログラミング言語の意味論が明確であることがまず第一に要求される。しかし一方で、プログラミング言語の記述力をより高めたい、よりパワフルにしたいという要求が存在することも明白で、それを満足させるための機能拡張が言語の意味論を不透明なものにしてしまうこともしばしばある。プログラミング言語の透明性や簡潔性を維持したままその記述力を拡張できるという能力は、プログラミング言語に期待される重要な能力であると言うことができる。リフレクションは、この要求を満足させるための有力な候補の一つである。

リフレクションは、Smithによる3-lisp[2]をその起源として、その後いくつかの言語に取り入れられてきた。しかしそれらの多くは、ベースとなる言語の意味自体が明確でないこともあって、意味論が不透明でありアドホックな印象を否めない。すなわち、現在の段階ではリフレクションは単なるプログラミング技術のレベルにあるといえるだろう。本稿では、リフレクションをプログラミング技術から理論的にも耐えうる概念とすることを目的として、リフレクティブな言語の意味論を論じることを試みる。そのために、逐次論理型言語であるPrologに注目し、リフレクティブなPrologを形式化する。

Prologにリフレクションを組み入れる意義は何か。これには、次のように答えることができるだろう。Prologが非常に多くの言語要素を用意しているため、その宣言的意味はもちろんのこと手続き的な意味も不明瞭になりがちだということである。前節で述べたように、リフレクションの機構を導入することによって少ないプリミティブで言語を構成することができる。これは言語の意味論を明確にするためにも重要なことである。またリフレクションを理論的に論じることにより、Prologにおける重要なプログラミング技法であるメタプログラミングの理論的枠組を与えることもできる。

本稿では、まずリフレクションについて論じる基礎として、次節でPrologとその意味論を形式化する。そして、第3節においてリフレクティブなPrologとしてのR-Prologを提案し、その意味論を与える。そして、最後にまとめと今後の展望について述べる。

2 Prologの形式化

この節では、Prologにリフレクションを組み入れるための基礎として、Prologの構文と意味論を形式化する。この形式化されたPrologをF-Prologと呼ぶ。

2.1 F-Prologの構文

定義 2.1.1 F-Prologは次の記号から構成される。

1. 可算個の変数。
   $X_1, X_2, \ldots, X_i, \ldots$;
2. 有限個の定数記号(ただし、特殊な定数記号としてnilを持つ)。
   $nil, c_1, c_2, \ldots, c_l$;
3. 有限個の関数記号(ただし、特殊な関数記号として$cons$を持つ)。
   $cons, f_1, f_2, \ldots, f_m$;
4. 有限個の述語記号。
   $p_1, p_2, \cdots, p_n$;
5. 区切り記号
   カンマ(,), ピリオド(.), 含意記号($\leftarrow$). □

Vを変数の集合、C, F, Pをそれぞれ定数記号、関数記号、述語記号の有限集合とする。述語記号と関数記号にはそれぞれ唯一の非負整数が対応付けられており、その整数を対応する述語(または関数)のアリティと呼ぶ。アリティnの述語(または関数)記号をn項述語(関数)記号と呼ぶ。アリティを明示的に表すため、n項述語記号pをp/nと書く場合がある。ただし、$cons$のアリティは2である。

定義 2.1.2 F-Prologの項、アトム、節は以下のように帰納的に定義される。

1. 項
   (a) 定数記号、及び変数記号は項である。
   (b) リストは項である。
   (c) fをn項関数記号、$t_1 \ldots t_n$を項とする時、$f(t_1, \ldots, t_n)$も項である。
2. アトム
   (a) pをn項述語記号とし、$t_1 \ldots t_n$を項とする時、$p(t_1, \ldots, t_n)$はアトムである。
3. 節
   (a) $a, a_1, \ldots, a_k$をアトムとするとき、$a \leftarrow a_1, \ldots, a_k$. ($k$は非負整数)は節である。ここで、$a$をヘッド、$a_1, \ldots, a_k$をボディと呼ぶ。また、$k$が0のときは特に単位節と呼び、$a \leftarrow$あるいは$a.$と書く。 □

Tを項の集合、ATをアトムの集合、CLを節の集合とする。また、変数を持たない項を閉項と呼ぶ。

ここで、関数記号 $cons$ と定数記号 nil に関して次のような慣習を用いる。t_1 と t_2 を項とするとき、項 $cons(t_1, t_2)$ をリストと呼び $[t_1|t_2]$と書く。$t_2 = nil$ の場合には特に$[t_1]$と書く。t_2 がさらにリスト $[l]$ である場合には、これを $[t_1, l]$ と書く。すなわち、$[t_1, \ldots, t_n] = [t_1|[t_2, \ldots, t_n]] = [t_1|[t_2|[\ldots[t_n|nil]\ldots]]]$である。また、$nil$ 自身は [] とも書かれる。

以下の定義は、プログラムの概念を定義するための準備である。

定義 2.1.3 clを節とするとき、clのヘッドの述語を節clの述語と呼ぶ。 □

節clの述語がpであることを強調するために、cl^pと書くことがある。

定義 2.1.4 変数の集合Vの有限部分集合から項の集合Tへの関数を代入と呼ぶ。すべての代入の集合を$Subst$と書く。 □

代入を、$\sigma, \sigma_1, \sigma_2, \ldots$の記号で表す。特に、恒等関数である代入を ϵ で表す。$\{X_1, \ldots, X_n\}$を定義域とする代入 σ に対して、$\sigma(X_i) = t_i (1 \leq i \leq n)$ であるとき、$\sigma = \{X_1/t_1, \ldots, X_n/t_n\}$ と書く。$\sigma = \{X_1/t_1, \ldots, X_n/t_n\}$ で、かつ $t_1, \ldots, t_n$ がすべて閉項であるとき、σ を閉代入と呼ぶ。さらに代入の概念は、項の集合Tを定義域とするように自然な拡張を行うことができる。以下では、特に断らない限り代入と言えばこの拡張された代入を指すことにする。また、項 t に代入 σ を作用させたときの結果を$t\sigma$ と書く。

定義 2.1.5 $\sigma_1 = \{X_1/t_1, \ldots, X_n/t_n\}$ と $\sigma_2 = \{Y_1/s_1, \ldots, Y_m/s_m\}$を代入とするとき、$\sigma_1$ と σ_2 の合成 $\sigma_1 \cdot \sigma_2$ は$\{X_1/t_1\sigma_2, \ldots, X_n/t_n\sigma_2, Y_1/s_1, \ldots, Y_m/s_m\}$ からある X_i に対して $Y_j = X_i$ となる Y_j/s_j と $X_i = t_i\sigma_2$ となる$X_i/t_i\sigma_2$ を除いたものである。 □

さらに、項の間の一般性に基づく順序関係やユニフィケーションの概念も通常の論理プログラムの理論と同様に定義される(例えば[1]参照)。さて、次にプログラムの概念を定義する。

定義 2.1.6 Cを節の有限集合とする。C上に順序関係を定義することで得られる有限順序集合 $P = (C, \leq)$ が次の条件を満たすとき、Pをプログラムと呼ぶ。

任意の2つの節$cl_1, cl_2 \in C$に対して

cl_1とcl_2の述語が等しい $\Longleftrightarrow$ $cl_1 \leq cl_2$または$cl_2 \leq cl_1$.

□

すなわち、プログラムとは節の有限集合であり、同じ述語を持つ節からなる部分集合の上に線形順序が定義されているものである。この順序は実際のPrologプログラムが持つ節の間の順序関係をモデル化するものである。プログラム全体の集合を$Prog$で表す。

$C = \{cl_1^{p_1}, \ldots, cl_{k_{p_1}}^{p_1}, \ldots, cl_1^{p_n}, \ldots, cl_{k_{p_n}}^{p_n}\}$ であり、各 p_i について $cl_1^{p_i} \leq cl_2^{p_i} \leq \cdots \leq cl_{k_{p_i}}^{p_i}$ と順序付けられているとき、P を $\{\langle cl_1^{p_1}, \cdots, cl_{k_{p_1}}^{p_1}\rangle, \ldots, \langle cl_1^{p_n}, \cdots, cl_{k_{p_n}}^{p_n}\rangle\}$ と書く。

定義 2.1.7 アトムの(空列も許す)有限列をゴールと呼ぶ。 □

$a_1, \ldots, a_n$ をアトムとするとき、これらからなるゴールを$\langle a_1, \ldots, a_n\rangle$ と書く。空列は、$\langle\rangle$ と書く。ゴールの集合を $Goal$ で表す。

2.2 F-Prologの意味論

この節では、F-Prologの意味を最小不動点として定義する。そのための準備として、まず必要な集合を定義することにしよう。まず、非負整数全体の集合をωで表す。集合 S に対して S^* は S の元の有限列全体の集合を、S^ω は S の元の無限列全体の集合を指すものとする。また、S の元 $s_1, \ldots, s_n$ からなる有限列を$\langle s_1, \ldots, s_n\rangle$、$s_1, \ldots, s_n, \ldots$ からなる無限列を$\langle s_1, \ldots, s_n, \ldots\rangle$、空列を $\langle\rangle$ で表す。列の長さを与える関数として ln があるとする。すなわち、$ln(\langle\rangle) = 0$、$ln(\langle s_1, \ldots, s_n\rangle) = n$、$ln(\langle s_1, \ldots, s_n, \ldots\rangle) = \omega$ である。さらに、S^* の元 l_1 と S^* または S^ω の元 l_2 の連接を$l_1 \cdot l_2$ で表す。すなわち、$l_1 = \langle s_1, \ldots, s_n\rangle$、$l_2 = \langle r_1, \ldots, r_m, \ldots\rangle$に対して $l_1 \cdot l_2 = \langle s_1, \ldots, s_n, r_1, \ldots, r_m, \ldots\rangle$である。$S^* \cup S^\omega$ の元 l_1 と l_2 の共通部分とは、$S^* \cup S^\omega$ の元 l であり、$l_1', l_2' \in S^* \cup S^\omega$ が存在して$l_1 = l \cdot l_1'$、$l_2 = l \cdot l_2'$ が成り立つものである。$S^* \cup S^\omega$ の元 l_1 と l_2 の共通部分の中で最長のものを$intersec(l_1, l_2)$ と書く。

まず、F-Prologの計算の状態を記述するための要素として、木の定義を行う。

定義 2.2.1 以下の条件を満たす ω^* の部分集合 T を木と呼ぶ。

1. 任意の ω^* の元 l と ω の元 n に対して $l \cdot \langle n\rangle$ がT の元であれば、$m < n$ なる任意の $m \in \omega$ に対して l および $l \cdot \langle m\rangle$は T の元である。
2. 任意の $l \in T$ に対して、$\{n | l \cdot \langle n\rangle \in T\}$ は有限集合である。 □

すべての木からなる集合を$Tree$で表す。また、その集合としての基数が有限である木を有限木、基数が無限である木を無限木という。木 T の元 l について、どんな $n \in \omega$ に対しても $l \cdot \langle n\rangle \notin T$ であるとき、l を T の葉と呼ぶ。また、$\langle\rangle$ を根と呼ぶ。

定義 2.2.2 木Tの元 l_1 と l_2 の間に次のような順序 $\leq_T$ を定義する。ただし、$l = intersec(l_1, l_2)$、$l_1 = l \cdot l_1'$、$l_2 = l \cdot l_2'$ とする。

$l_1 \leq_T l_2 \iff l_2' = \langle\rangle$であるか、$l_1' = \langle n\rangle \cdot l_1^+$ かつ $l_2' = \langle m\rangle \cdot l_2^+$ かつ $n < m$

□

定義 2.2.3 T_1 および T_2 を木とし、さらに l を T_1 の葉とする。このとき、T_2 の T_1 への l における連接 $cat(T_1, l, T_2)$は次のように定義される。

$$cat(T_1, l, T_2) = T_1 \cup \{l \cdot m | m \in T_2\}$$ □

すなわち、$cat(T_1, l, T_2)$ は T_1 の葉 l に T_2 の根を継ぎ足して得られる木である。

次に代入ストリームの概念を定義しよう。

定義 2.2.4 $\langle\sigma_1, \ldots, \sigma_i, \ldots\rangle \in (Subst \cup \{\bot\})^\omega$ が代入ストリームであるとは、それが以下の条件を満たすことである。

ある i に対して $\sigma_i = \bot$ であるならば、$k \leq i$ が存在して、任意の $j \leq k$ に対しては $\sigma_j \neq \bot$、任意の $j > k$ に対しては$\sigma_j = \bot$ が成り立つ。 □

代入ストリーム $\langle\sigma_1, \ldots, \sigma_k, \bot, \bot, \ldots\rangle$は、省略型として $\langle\sigma_1, \ldots, \sigma_k, \bot\rangle$ と書かれる。代入ストリームの集合を SS で表す。

補題 2.2.1 代入ストリームの集合 SS は次のように定義される順序 $\leq_{SS}$ によって、完備順序集合になる。s_1、s_2を SS の元、$s = intersec(s_1, s_2)$ とし、$s_1 = s \cdot s_1'$、$s_2 = s \cdot s_2'$ とする。

$s_1 \leq_{SS} s_2 \iff s_1 = \langle\bot, \ldots\rangle$

証明. 定義より明らか。 □

F-Prologの意味を記述するために必要な次の概念は、代入ストリーム付きゴールである。

定義 2.2.5 $a_1, \ldots, a_n$ をアトム、$s_1, \ldots, s_n$ を代入ストリームとするとき、代入ストリーム付きゴール gs は以下の形をした有限列である。

$gs = \langle a_1, s_1, \ldots, a_n, s_n\rangle$ □

代入ストリーム付きゴールの集合を $SGoal$ で表す。$SGoal$ も、SS 上の順序を自然に拡張して、完備順序集合とすることができることを注意しておく。

ここで、代入ストリーム、代入ストリーム付きゴールに関連した演算を定義しておく。

定義 2.2.6 1. $s = \langle \sigma_1, \ldots, \sigma_n, \bot \rangle$ を代入ストリーム、 σを代入とするとき、 $s * \sigma = \langle \sigma, \ldots, \sigma_n, \sigma, \bot \rangle$。
4. $s = \langle \sigma_1, \ldots, \sigma_n, \bot \rangle$ を代入ストリームとするとき、 $trancate(s) = \langle \sigma_1, \ldots, \sigma_n \rangle$ 。
3. $gs = \langle a_1, s_1, \ldots, a_n, s_n \rangle$ を代入ストリーム付きゴール、 σ を代入とするとき、 $i(1 \leq i \leq n)$ に対して、 $gs * \langle i, \sigma \rangle = \langle a_1, s_1, \ldots, a_i, s_i * \sigma, \ldots, a_n, s_n \rangle$ 。
4. $g = \langle a_1, \ldots, a_n \rangle$ をゴールとするとき、 $initialize(g) = \langle a_1, \langle \bot \rangle, \ldots, a_n, \langle \bot \rangle \rangle$ と定義し、初期代入ストリーム付きゴールと呼ぶ。
5. $gs = \langle a_1, s_1, \ldots, a_n, s_n \rangle$ を代入ストリーム付きゴールとするとき、 $ans(gs) = s_n$ である。 □

次に探索木の概念を定義する。それに先立ち、まず代入と節の対からなる有限列の集合 $SCL = (Subst \times CL)^*$ を定義しておく。

定義 2.2.7 ゴール g の探索木 τ_g とは、以下の条件を満たす木 $T \in Tree$ から $SCL \cup SGoal \cup AT$ への関数である。

1. $l = \langle\rangle \in T$ に対しては、$\tau(l) = initialize(g) \in SGoal$。
2. 任意の $l \cdot \langle m \rangle \in T$ に対して $\tau(l) \in SGoal$であれば、 $\tau(l \cdot \langle m \rangle) \in AT$ である。
3. 任意の $l \cdot \langle m \rangle \in T$ に対して $\tau(l) \in SAT$であれば、$\tau(l \cdot \langle m \rangle) \in SCL$ である。
4. 任意の $l \cdot \langle m \rangle \in T$ に対して $\tau(l) \in SCL$であれば、 $\tau(l \cdot \langle m \rangle) \in SGoal$ である。 □

ゴールgの探索木全体の集合を$STree_g$とする。また、$dom(\tau_g)$によってτ_gの定義域を表す。さらに、文脈より明らかな場合には、ゴールを表す添字を省略することがある。

定義より、任意の探索木 τ 、任意の $l \in dom(\tau)$ に対して、 $ln(l)$の 3 による除算の剰余が 0 であれば $\tau(l) \in SGoal$ 、 1 であれば $\tau(l) \in AT$、 2 であれば $\tau(l) \in SCL$ であることが分かる。これらの場合をそれぞれ、$SGoal$ ノード、 AT ノード、 SCL ノードという。

定理 2.2.2 ゴール g の探索木全体の集合 $STree_g$ は、以下のように定義される順序$\leq_{STree}$ によって完備順序集合になる。ここで、τ、τ' は$STree_g$の任意の元である。

$\tau \leq_{STree} \tau' \iff dom(\tau) \subseteq dom(\tau')$、かつ任意の $l \in dom(\tau)$ に対して l が AT ノード、または SCL ノードのときは $\tau'(l) = \tau(l)$、 $SGoal$ ノードのときは $ln(\tau(l)) = ln(\tau'(l))$ かつ $a_i = a'_i$、 $s_i \leq_{SS} s'_i$ 。ただし、$\tau(l) = \langle a_1, s_1, \ldots, a_n, s_n \rangle$ 、$\tau'(l) = \langle a'_1, s'_1, \ldots, a'_n, s'_n \rangle$ であるとする。

証明. 定義された$\leq_{STree}$によって$STree_g$ が順序集合になることは明らかである。$Stree_g$ が完備順序集合であることを証明するためには、次の 2 つを示せばよい。

1. $Stree_g$ が最小元を持つ。
2. $Stree_g$ の任意の鎖が極限を持つ。

1. については $\tau_0 = \{< \langle\rangle, intitialize(g) >\}$の存在によって満たされているので 2. を示せば十分。

$\{\tau_i | i \in \omega\}$ を任意の鎖とする。ここで、$\hat{\tau}$を次のように定義する。まず、$dom(\hat{\tau}) = \bigcup\{dom(\tau_i) | i \in \omega\}$と置く。すると、$dom(\hat{\tau})$ の任意の元 l に対して、ある $n \in \omega$ が存在し$l \in dom\tau_i$ となる。 l が AT ノードか SCL ノードであるときは、このn を用いて $\hat{\tau}(l) = \tau_n(l)$ と定義する。 l が $SGoal$ ノードのときは、 $SGoal$ における鎖 $\{\tau_i(l) | i \geq n\}$ の極限を取り、$\hat{\tau}(l) = \sqcup\{\tau_i(l) | i \geq n\}$ と定義する。この定義が well-defined であり、定義された $\hat{\tau}$ が 鎖の極限となっていることは明らかであろう。 □

定義 2.2.8 探索木全体からなる完備順序集合 $STree$ を次のように定義する。

$STree = \biguplus\{STree_g | g \in Goal\}$

ただし、 $\biguplus\{STree_g | g \in Goal\}$ は $\{STree_g | g \in Goal\}$ の順序集合としての直和で、新たな最小元として $\{\bot\}$ が加わったものである。 □

定義 2.2.9 $Prog^+$ 、 $Goal^+$ をそれぞれ $Prog$ 、 $Goal$ に最小元 $\bot$ を加えて得られる完備順序集合とする。このとき状態の集合 $State$ は次のように定義される。

$State = \{\phi \in ((Prog^+ \otimes Goal^+) \to STree) |$ 任意の $P \in Prog, g \in Goal$に対して$\phi(P, g) \in STree_g\}$

ただし、$(Prog^+ \otimes Goal^+ \to STree)$ は $Prog^+ \otimes Goal^+$ から$STree$ への連続関数からなる完備順序集合である。 □

定理 2.2.3 $State$ は次のように定義される順序 $\leq_{State}$ によって完備順序集合になる。st_1 、 st_2 を $State$ の任意の元とする。

$st_1 \leq_{State} st_2 \iff$ 任意の $P \in Prog^+$、 $g \in Goal$ に対して$st_1(P) \leq st_2(P)$.

証明. 定義より明らか。 □

定義 2.2.10 関数 $\mathcal{U} : AT \times Prog \to SCL$ を次のように定義する。ただし、aをアトム、Pをプログラムとする。

$\mathcal{U}(a, P) = \langle (\sigma_1, cl_{k_1}\sigma_1), \ldots, (\sigma_n, cl_{k_n}\sigma_n) \rangle$

ここで、$cl_{k_1},\ldots,cl_{k_n}$ はPにおいて a とユニファイ可能な節で、$\sigma_1,\ldots,\sigma_n$は対応する mgu である。ただし、Pにおいて$cl_{k_1} \leq \ldots \leq cl_{k_n}$ という関係にあるとする。aとユニファイ可能な節がないときは、$\langle\rangle$を返す。 □

定義 2.2.11 関数 $\mathcal{E} : State \rightarrow State$ を以下のように定義する。

$\phi \in State$、$P \in Prog$、$g \in Goal$ とする。まず、$\phi = \perp$ のとき、$\mathcal{E}(\phi)(P,g) = initialize(g)$ と定義する。次に $\phi(P,g) = \tau$ とするとき、$\mathcal{E}(\phi)(P,g) = \tau'$ なる τ' を以下のアルゴリズムによって構成する。

Algorithm 1.

$l_1, l_2, \ldots, l_i, \ldots$ を $dom(\tau)$ の元の数え上げとする。

1. $\tau' := \tau$, $i := 0$ とおく.
2. $i := i+1$.
3. if $i > |dom(\tau)|$ then goto End: .
4. case(l_i)
   - l_i が $SGoal$ ノードで, かつ $dom(\tau)$ の葉である場合.
     (a) $\tau(l_i) = \langle a_1, s_1, \ldots, a_n, s_n\rangle \neq \langle\rangle$であるとき.
     $\tau' := \tau' \cup \{< l_i \cdot \langle 1\rangle, a_1 >, \ldots, < l_i \cdot \langle n\rangle, a_n >\}$.
     (b) $\tau(l_i) = \langle\rangle$ であるとき.
     $l_i = l \cdot \langle n_1, n_2, n_3\rangle$ かつ, $\langle \ldots, (\sigma_{n_3}, cl_{n_3}), \ldots\rangle = \tau(l \cdot \langle n_1, n_2\rangle)$ であるとすると,
     $\tau' := \tau' - \{< l, \tau(l) >\} \cup \{< l, \tau(l) * \langle n_1, \sigma_{n_3}\rangle >\}$.
   - l_i が SCL ノードで, かつ $dom(\tau)$ の葉である場合.
     (a) $\tau(l_i) = \langle(\sigma_1, cl_1), \ldots, (\sigma_n, cl_n)\rangle$ であるとき.
     $\tau' := \tau' \cup \{< l_i \cdot \langle k\rangle, initialize(body(cl_k)) > | 1 \leq i \leq n\}$.
     ただし、$body(cl)$ は節 cl に対してそのボディを与える関数である。
     (b) $\tau(l_i) = \langle\rangle$ のとき.
     $l_i = l \cdot \langle n_1, n_2\rangle$ とすると,
     $\tau' := \tau' - \{< l, \tau(l) >\} \cup \{< l, truncate(\tau(l)) >\}$.
   - l_i が AT ノードである場合.
     $l_i = l \cdot \langle k\rangle$ とする。
     (a) $k = 1$ の場合.
     l_i が葉であるときは,
     $\tau' := \tau' \cup \{< l_i \cdot \langle 1\rangle, \mathcal{U}(\tau(l_i), P) >\}$.
     そうでないときは, $\tau' := \tau'$.
     (b) $k \neq 1$ の場合.
     $|\{j | l_i \cdot \langle j\rangle \in dom(\tau)\}| = m$, $\tau(l) = \langle a_1, s_1, \ldots, a_n, s_n\rangle$ とする. ここで$s_{k-1} = \langle \sigma_1, \ldots, \sigma_h, \perp\rangle$, かつ $h > m$ であれば, $\tau' := \tau' \cup \{< l_i \cdot \langle j\rangle, \mathcal{U}(\tau(l_i)\sigma_j, P) > | m+1 \leq j \leq h\}$.
   - otherwise.
     $\tau' := \tau'$.
5. $l_i = l \cdot \langle n_1, n_2, n_3\rangle$ が $SGoal$ ノードで、かつ $l_i \neq \langle\rangle$ であり, $n_3 = 1$ のときか あるいは $n_3 \neq 1$ でかつ $ans(\tau'(l \cdot \langle n_1, n_2, n_3 - 1\rangle)) = \langle \sigma'_1, \ldots, sigma'_m\rangle$ であるとき .
   $\tau'(l) = \langle a_1, s_1, \ldots, a_n, s_n\rangle$, $\tau'(l \cdot \langle n_1, n_2\rangle) = \langle \ldots, (\sigma_{n_3}, cl_{n_3}), \ldots\rangle$, $\tau'(l_i) = \langle a'_1, s'_1, \ldots, a'_m, s'_m\rangle$ とする.
   $\tau' := \tau' - \{< l, \tau'(l) >\} \cup \{< l, \langle \ldots, a_{n_1}, s'_n, a_{n_1+1}, \ldots\rangle >\}$.
6. goto 2.

End: □

この $\mathcal{E}$ が F-prolog の実行を模倣している関数である。さて、この $\mathcal{E}$について次の重要な性質を示すことができる。

定理 2.2.4 関数 $\mathcal{E} : State \rightarrow State$ は連続である。すなわち、$State$ 上の任意の鎖$\{st_i | i \in \omega\}$ (ただし、$st_1 \leq_{State} st_2 \leq_{State} \cdots \leq_{State} st_i \leq_{State} \cdots$)に対して、
$\mathcal{E}(\bigsqcup\{st_i | i \in \omega\}) = \bigsqcup\{\mathcal{E}(st_i) | i \in \omega\}$
が成立する。

証明. 略。 □

完備順序集合上の連続関数は最小不動点を持つことが知られているので$\mathcal{E}$が最小不動点を持つことが示されたことになる。これを、$lfp(\mathcal{E})$と置こう。

定義 2.2.12 $P \in Prog$ 、$g = \langle a_1, \ldots, a_n\rangle \in Goal$ とおく。このとき、
1. $\tau^f_{(P,g)} = lfp(\mathcal{E})(P,g)$
   と定義する。
2. $ans(\tau^f_{(P,g)}(\langle\rangle)) \in SS$ を P におけるg の回答と呼ぶ。

□

ここまで述べてきた F-Prolog の意味論に関して、いくつかの性質を示すことができる。例えば、任意の述語記号 p/n に対して、プログラム P におけるゴール$(p(X_1,\ldots,X_n))$ の回答を与えることは、P の S-model を与えることと同値になる。さらに、この意味論では節の間の順序関係を考慮に入れており、それによって答として与えられる代入の間に順序関係が入っている。これは、Prolog の逐次的な実行を形式化したものであり、その意味でここで与えた意味論は宣言的意味論と手続き的意味論の中間的な役割を果たすものだと言える。

3 R-Prolog

この節では、前節で形式化した F-Prolog にリフレクションの概念を導入する。このリフレクティブな Prolog を R-Prolog と呼ぶ。これによって、リフレクションを形式的に捉えることが可能になり、理論的考察のための基礎を提供することができる。

3.1 R-Prolog の構文

まず、R-Prolog の構文を定義する。

定義 3.1.1 R-Prolog の記号は定義 2.1.1で述べた F-Prolog の記号に次のものが加わる。

1. 特殊な 1 項述語記号、reflect。
2. 特殊記号、↑(quote)、↓(unquote)。

□

定義 3.1.2 R-Prolog の項、アトム、節は定義 2.1.2の F-Prolog と同様に定義されるが、新たに加わった記号に対して次の項目が加わる。

1. quote 記号。
   (a) 変数 X、定数記号 c、関数記号 f、述語記号 p に対して、$\uparrow X$、$\uparrow c$、$\uparrow f$、$\uparrow p$ は新たな定数記号である。
   (b) 項 $f(t_1,\ldots,t_n)$ に対して、$\uparrow f(t_1,\ldots,t_n) = [\uparrow f, \uparrow t_1,\ldots,\uparrow t_n]$。
   (c) アトム $p(t_1,\ldots,t_n)$ に対して、$\uparrow p(t_1,\ldots,t_n) = [\uparrow p, \uparrow t_1,\ldots,\uparrow t_n]$。
   (d) 節 $a \leftarrow a_1,\ldots,a_n$ に対して、$\uparrow (a \leftarrow a_1,\ldots,a_n) = [\uparrow a, \uparrow a_1,\ldots,\uparrow a_n]$。

   これらの形式の項を、特に<u>quote 形式</u>という。
2. unquote 記号。
   tを quote 形式の項であるとすると、$\downarrow t$ は t から ↑(quote) を外したものである。

□

さらに、項や節の順序対であるとか有限列などの概念も、リストによって項として表現することが可能である。したがって、プログラム、ゴール、代入などの構文的対象はもちろん、代入ストリームや有限探索木のような意味論的対象も項として R-Prolog の言語内で表現できることに注意しよう。

リフレクションは、計算の状態を対象化しプログラムレベルからそれにアクセスできるような機構である。計算の状態は、各瞬間においては有限の対象であり、したがって R-Prolog の項として表現できる。R-Prolog においてリフレクションが記述できるのは、まさにこの事実によるのである。

3.2 意味論

R-Prolog の意味論を形式化するために、2.2 節で定義したような意味論的対象、すなわち代入ストリームや状態の集合を定義しなければならない。しかし、これらは定義 3.1.1 および定義 3.1.2 で定義された拡張された言語の下で、再定義をして行くことによって得ることができる。こうして得られた R-Prolog の状態の集合を $State^R$ と書くことにする。

定義 3.2.1 関数 $\mathcal{E}^R : State^R \rightarrow State^R$ は、定義 2.2.11 に以下の部分が追加されたものとして定義される。

4′. case(l_i)

- l_i であるとき.
  (c) $\tau'(l_i) = p(t_1,\ldots,t_n)$ の述語 p がリフレクト述語であるとき.
  $\tau' := \tau' \cup \{< l \cdot \langle 1\rangle, initialize(\tau'(l_i)\&(\uparrow P, \uparrow \tau') >\}$
  ただし、$\tau'(l_i)\&(\uparrow P, \uparrow \tau') = p(t_1,\ldots,t_n, \uparrow P, \uparrow \tau')$ である。

□

$\mathcal{E}^R$ についても、定理 2.2.4 と同様の定理が成り立ち、連続関数であることが期待される。この事実の証明は、別の機会に譲ることにする。

リフレクティブな Prolog の意味は、まさにこの $\mathcal{E}^R$ によって表現されているのである。

4 おわりに

本稿では、論理型言語である Prolog の意味を代入ストリームに基づく最小不動点によって定義し、そ

れに基づいてリフレクションの機構を持たせた R-Prolog を提案した。そして、同じ枠組の下で R-Prolog の意味論を形式化することを試みた。これによってリフレクションのメカニズムに理論的な枠組を提供することが可能になり、さらに従来取り扱いが厄介であった Prolog におけるメタロジカルな述語、例えば cut や assert、retract 等の意味を明確にすることを可能にした。

今後の課題としては、第一に理論的考察を深めることがあげられる。それによってリフレクティブなプログラムの正当性や同値性などの議論を形式的に行うことができる。第二には、ここで提案した代入ストリームの概念を発展させ、他の論理型言語、例えばGHC等、におけるリフレクションを考察することである。第三にあげられることは、R-Prologの実現を行うことによって、リフレクションやメタプログラミングの応用分野を開拓することである。さらに、知識獲得や学習、非単調推論に対するリフレクションからの、実現と形式化という興味深い課題も残されている。

謝　辞

本研究を進めるにあたり、貴重な時間を割いて議論に付き合って下さり、有益な助言を下さった国藤室長、田中(二)、神田研究員に感謝致します。

参考文献

[1] J. W. Lloyd. *Foundations of Logic Programming.* Springer, 1984.

[2] B. C. Smith. Reflection and semantics in lisp. In *Proc. 11th ACM Symposium on Principles of Programming Languages*, pages 23–35, 1984.