ソフトウェア基礎論　18−1
（１９８６・９・３０）

PROLOG述語（呼び出し）の決定性について

沢村　一

富士通㈱国際情報社会科学研究所

本論文では、a-決定性とr-決定性と呼ばれる二つの決定可能な述語呼び出しの部分クラスを導入し、同値であるがそれらとは異なった定義法及びそれらの性質、拡張法等について述べる。これらの概念はPrologのプログラマが、述語（呼び出し）の決定性に関して日常無意識に用いている直観を自然に、そして統語論的に定義したものである。さらに、述語（呼び出し）の決定性の検出の意義、利用等を二つの面から論じる。一つはPrologプログラムの最適化という面からであり、もう一つはプログラムの信頼性に関わるプログラムの性質という面からである。

On the Determinacy of a Prolog Predicate (Call)

Hajime SAWAMURA

International Institute for Advanced Study of Social Information Science
(IIAS-SIS), FUJITSU LIMITED, 140 Miyamoto, Numazu, Shizuoka 410-03 Japan

Two kinds of determinacy, called a-determinacy and r-determinacy, are formally defined. Their properties and extensions are then explored. Our definitions capture the intuitions about Prolog programs many experienced programmars may have.

The significance of detecting deterministic computations and its utilization are also discussed from two aspects; one from source-to-source transformation (optimization) of Prolog programs and the other from a program property concerned with the reliability factor of Prolog programs.

1. はじめに

「決定性」、「非決定性」という言葉は、計算機科学ではオートマトン・言語理論、計算理論、プログラム言語理論及び言語の意味論・検証論など、また他の科学においてもしばしば現れる。これらの言葉の意味はそれぞれの分野毎に少しずつ異なっているようであるが、決定性、非決定性自身を検出することが必要とされる問題は少ないように思われる。本論文では、非決定性プログラミング言語であるProlog〔1〕の世界で、述語（呼び出し）の決定性の検出が重要となる問題を論じる。

これまで、非決定的な性格をもつ手続きを表現することを可能にする非決定性プログラム言語はいく人かの人達によって研究されてきた。その中でも、Floyd〔2〕、Dijkstra〔3〕による言語、人工知能向き言語であるMicro-planner〔4〕は、最近のProlog、Concurrent Prolog〔5〕に加えてよく知られている非決定性プログラム言語である。

これらの言語で書かれたプログラムは二つの主要な意味で非決定的である。すなわち、Don't care (DC) と Don't know (DK) である〔6〕。PrologとMicro-planner は後者の非決定性を実現し、Concurrent PrologとDijkstraのガード付命令による非決定性言語は前者の非決定性を実現しているといえる。Floyd の言語は両方の解釈をもち得る。

我々は論文〔7〕の中で、DC非決定性もDK非決定性も共にそれらの決定問題が決定不能であることを示した。DC決定性の決定不能性は容易に一階論理の妥当性判定問題に帰着することによって示されるし、またDK非決定性の決定不能性の証明はRussell のパラドックスの如く、Prologプログラムによって二律背反命題を作ることによって与えられる。決定性の判定問題が決定不能である以上、次に我々が求めなければならないのは、いかにして広い範囲の決定可能な決定性述語（呼び出し）のクラスを定義するかということになる。論文の前半では、a-決定性とr-決定性と呼ばれる二つの決定可能な述語呼び出しの部分クラスを導入し、同値であるがそれらとは異なった定義法及びそれらの性質を調べる。論文の後半においては、述語（呼び出し）の決定性の検出の意義を、現実のPrologプログラムの最適化とPrologプログラミング方法論等の観点から論じる。

2. 述語の決定性

2. 1 記号規約

ここでは、Prologの構文法とその実行的意味に関する知識を仮定する〔1〕。そして、以下の議論に特有の記法を定める。

Prologプログラムは次の形式をした順序づけられた節の有限集合である。

$$H :- B1, \cdots, Bn. \quad (n \geq 0)$$

ここで、H は節のヘッド、各Biは述語呼び出しあるいはゴールと呼ばれる。" B1, ・・・, Bn "は節の本体と呼ばれる。ゴール（述語名）が与えられたとき、そのゴール（述語名）とヘッドが同じ名前とアリティをもつ節の順序付の集まりは、そのゴール（述語名）の述語定義体とか、そのゴール（述語名）の定義節と呼ばれる。

以下、Prologの構文要素の上を動くメタ変数に対して次のような記号を用いることにする。

【記号規約】

(1) (添字付) 文字 P, G, H はゴールあるいはヘッドを表す。文字 p, q, rは述語名あるいは命題を表す。

(2) (添字付) ギリシャ文字 Γ、Δはゴールのコンマで区切られた列を表す（空列も含む）。Γが空列のときそれはゴール "true" と同一視する。文字A はPrologの項の非空な列を表す。

2. 2 述語（または、述語呼び出し（ゴール））の決定性

Prologの述語呼び出し（ゴール）が決定的であることを次のように定義する。

【述語呼び出しの決定性】

述語呼び出しが決定的であるとは、それが呼ばれたとき、その定義体の高々一つの節で成功し、バックトラックされたとき、決して成功することはないことをいう。

この定義は、次のようなゴールが決定的であることを含意することに注意されたい：

（i）最初に呼ばれたとき停止しないゴール

（ii）最初の実行で成功し、バックトラックしたとき停止しなくなるゴール

次にこの定義を箱による制御フローモデル〔8〕で示してみよう。このモデルでは、一つのゴールの非決定的な計算は、そのゴールへの制御の出入りを次のような腕が四本ある箱によって表現することによって捉えられる。

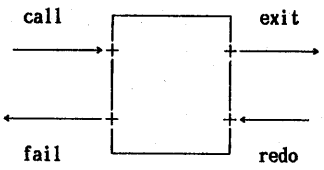

図１：制御のボックスモデル

また、Prologの質問及び定義節のボディのゴール列は脇が四本あるこのような箱がAND 結合で図２のように並んだものであると考えることができる。

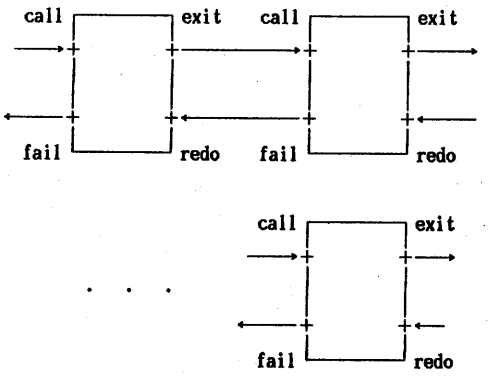

図２：ゴールのAND 結合

このとき、代替節が複数ある場合で、一つのゴールを決定的に達成する場合の制御の可能な流れの全体の様子は図３のように表現することができる。図３はゴールが決定的であることからその定義体は排他的にOR結合されていることを示しており、ノード⊕はバックトラックしてきた制御は左の方に流し、failした制御は下の方に流す働きをもつものである。もちろん、これは再帰的な図を表しているつもりである。

我々の定義と実質的に同じものは〔９〕、〔１０〕にも見出され、Prologにおける決定性、非決定性の意味として共通に了解されているものである。そして、これは、述語（関係）が関数的であるとする決定性の定義よりも一般的な定義である。すなわち、どの引数が入力で、どの引数が出力であるかということには一切関わらない定義である。

しかしながら、いずれの定義においても、一般に述語呼び出しが決定的であるか否かを決定することは決定不能であるので（その証明は〔７〕で与えられている）、本論文ではa-決定性とr-決定性と呼ばれる互いに関係する二つの決定可能な述語呼び出しの部分クラスを定義し同値であるがそれとは異なった定義及びそれらの性質について論ずる。

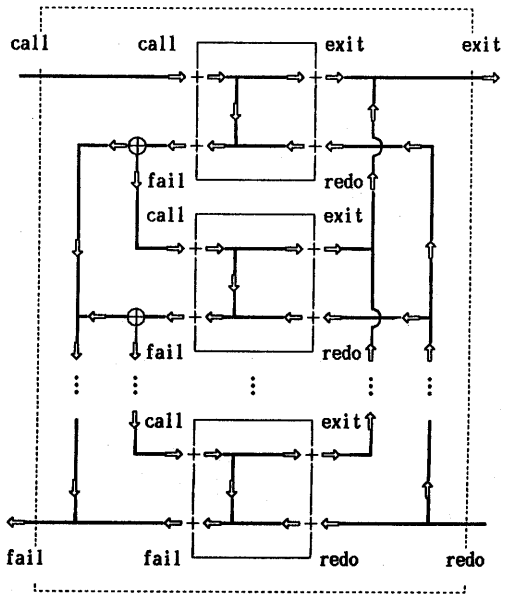

図３：決定性ゴールの制御フロー

一般に、述語呼び出しの成功、失敗はその引数の内容によって決まる。実際、以下で定義されるr-決定性という概念は述語の引数に依存した決定性を意味し、他方a-決定性という概念はその特別な場合として引数に全く依存しない決定性を意味するものである。

【ａ－決定性とｒ－決定性】

以下の定義において、プログラムを動的に変更する構成要素、例えば＂assert＂、＂retract＂等は関連する述語定義体には含まれないことを仮定する。

述語呼び出し p(A) が a- 決定的、あるいは r- 決定的であるということを、次のように相互帰納的に定義する。

(i) pが組み込みの述語名でかつp(A)が決定的であれば、p(A)は a- 決定的であり、かつ r- 決定的である

（ii）述語 p に関する定義体を次のものとする。

```
H1 :- Γ1.
 ⋮
Hi :- Γi, [!,] Δi.   (ここで、カット記
          号は存在するなら最右とする)
 ⋮
Hn :- Γn.
```

このとき、各Hi(1≦ i ≦n)に対して、次の条件のうち(1)あるいは(2)が成立するならば、p(A)は a- 決定的であり、p(A)と単一化可能なHiに対して(1)～(3)のいずれかが成立するならば、p(A)は r- 決定的である。

(1) Hi の本体にカットが存在し、Δi はすべて a- 決定的か、r-決定的である。

(2) Hi の本体にカットが存在せず、Hiは定義体の最後の節であり、Γi 、Δi はすべて a- 決定的か、r-決定的である。

(3) Hi の本体にカットが存在せず、p(A)はHj(i+1 ≦ j ≦n)のいずれとも単一化不可能、さらに、Γi 、Δi はすべて a- 決定的か、r-決定的である。

注意:

① プログラムが相互帰納的に定義されている場合でも決定的と判定されることがある。例えば、次の相互帰納的プログラム

```
q :-  Γ,!,r.
r :-  Θ,q, Δ,!.
```

では、呼び出しq 、r 共にa-決定的でもあり、r-決定的でもあることがわかる。より一般的には次のようにいうことができる。上の決定性の定義における述語 p の定義体

```
H1 :- Γ1.
 ⋮
Hi :- Γi, [!,] Δi.   (ここで、カット記
          号は存在するなら最右とする)
 ⋮
Hn :- Γn.
```

において、各 i (1≦ i ≦n)に対して条件(1)のΔi 、条件(2)及び(3)のΓi,Δi を述語p に関する述語定義体のクリティカルパートと呼ぶことにする。このとき、プログラムを構成する各述語定義体のクリティカルパートに含まれる述語呼び出し間のクロス・レファレンス(cross reference)を考えると（これは一般に有向グラフになる）、これにサイクルがないことが決定性判定のためには必要である。上の例では、クリティカルパート間のクロス・レファレンスには相互呼び出しは存在しない。

② プログラム（手続きの集まり）が与えられたとき、その中から上の定義を用いてa-決定的述語を抽出することができる。そしてこの抽出は一意に定まる。すなわち、どの述語からa-決定性の判定を行うかには依存しない。これは①のクリティカルパートのクロス・レファレンスの一意性による。

③ クリティカルパートに(G1;G2) 、(P -> G1;G2) のような形式の複合ゴールが含まれているとき、現在は簡単さのためにそれらは単に非決定的ゴールとして扱われている。(G1;G2) が決定的と判定されるためにはG1、G2共に決定的でかつそれらのゴールの成功が排他的であることが必要となる。

我々の定義の正当性は次の系によって保証される。

系 1. ゴール p(A) が a- 決定的か r- 決定的ならば、それは決定的である。

証明. 定義の構成に関する帰納法に従がう。(i) のケースは明らか。(ii)のケースは3 つの条件を一つ一つ調べれば十分であるが、それらはPrologの実行意味論と帰納法の仮定から直ちに導かれる。■

a-決定性とr-決定性の定義は三つの概念に基づいていた。すなわち、カット、組み込みの決定的述語及び静的に決定される単一化可能性である。別の言い方をすると、それらは述語呼出しに現れる変数がどのような型の引数を取るかということには関わっていない。すなわち述語の意味に関わることなく定められる決定性である。ここで、述語の意味とは、その述語が成功する項の領域をいう。

定義より、ゴールが a- 決定的ならば、それはその引数の形に依らずいつも決定的であることがわかる。別の言い方をするならば、a-決定的なゴールは、それがゴールの中の引数の形式に依存しないという意味で絶対的に決定的である(absolutely deterministic)。それゆえゴール p(A) が a- 決定的であることがわかったとき、述語 p を a- 決定的とか、あるいは単に決定的と呼ぶことがある。他方、絶対的な決定性とは対照的に、r-決定的なゴールは、その述語がどのように呼ばれるかに依存しているので相対的に決定的である (relatively deterministic)。これらの観察を次のように系として表明しておく。

系 2. ゴールが a- 決定的ならば、それはまた r-決定的である。

さらに、定義より、

系　3．組み込みの述語を除き、決定的述語呼出しがa-決定的と判定されないのは、その定義体の数が2より大きく、かつその中にカット記号が全く出現しないときである。

もちろん、次のプログラム例から見られるように系3の条件は強過ぎるように思われるかもしれない。

```
H :-  Γ, fail.
H :-  Δ.
```

ここでカット記号はΓ及びΔには出現せず、Δのすべての述語呼出しは a- 決定的か r- 決定的である。このような個々の場合を上の定義に組み込むことは難しくないが、それよりもここではできるだけ一般的な決定性の定義を設定することにより関心があった。

系　4．命題レベルでは、すなわち、調べようとしている述語に変数が含まれないとき、a-決定性は r- 決定性に一致する。

系　5．すべての項A に対してゴール　p(A) はr-決定的である　<=> 述語p はa-決定的である。

証明．(=>) a- 決定性とr-決定性の定義の条件(3)は、仮定より排除され、したがって述語p はa-決定的となる。(<=)　系 2より。■

補題　6．任意の引数列A に対して、ゴールp(A)が成功する（証明可能である）ことと、X を変数列とするときp(X)が成功すること（証明可能である）とは同値である。

証明：一階論理の次の定理による。

$\vdash \forall A.p(A) \Leftrightarrow \vdash p(X)$; X は自由変数　■

系　7．X を変数の列とする。ゴールp(X)はr-決定的である <=>　述語ｐはa-決定的である。

証明：系５と補題６による。■

系５と７は決定性に関して別の同値な定義を示唆する。すなわち、最初にr-決定性を定義し、次に系５あるいは系７にしたがってa-決定性を定義するものである。正確に表現すると、次のようになる。

【ｒ－決定性】

以下の定義において、プログラムを動的に変更する構成要素、例えば＂assert＂、＂retract＂等は関連する述語定義体には含まれないことを仮定する。

述語呼び出し p(A) が r- 決定的であるということを次のように帰納的に定義する。

(i) pが組み込みの述語名でかつp(A)が決定的であれば、p(A)はr-決定的である。

（ii）述語 p　に関する定義体を次のものとする。

```
H1 :- Γ1.
⋮
Hi :- Γi, [!,] Δi.   (ここで、カット記
                      号は存在するなら最右とする)
⋮
Hn :- Γn.
```

このとき、p(A)と単一化可能な各Hi(1≦ｉ≦n)に対して(1)～(3)のいずれかが成立するならば p(A) は r- 決定的である。

(1) Hi の本体にカットが存在し、Δi はすべてr-決定的である。

(2) Hi の本体にカットが存在せず、Hiは定義体の最後の節であり、Γｉ、Δi はすべてr-決定的である。

(3) Hi の本体にカットが存在せず、p(A)はHj(i+1≦ｊ≦n)のいずれとも単一化不可能、さらに、Γｉ、Δi はすべてr-決定的である。

【a-決定性$_1$】

述語ｐは、すべての項A に対してゴールp(A)がr-決定的であるとき、a-決定的と呼ばれる。

あるいは、単に、

【a-決定性$_2$】

述語ｐは、ゴールp(X)がr-決定的なとき、a-決定的と呼ばれる。

以上の定義は以前の定義より簡単なように思われるが、a-決定性$_1$ の定義は構成的でないことを注意しておかなければならない。

例　1．述語ｐは a- 決定的である。

```
p(a) :- write(a1), nl, !, write(a2).
p(b) :- write(b).
```

すなわち、ゴールp(X)はr-決定的である。

例　2．述語呼出しq([a, b]) は r- 決定的であるが、q(X)はそうではない。ここで、述語p の定義体は例 1のものとする。

```
q( [c] ) :- write(c), p(b).
q( [a |X ] ) :- write(a), p(X).
```

例　3．次の述語transform はa-決定的である。

```
transform('end-of-file') :- !.
transform(T) :- fold(T,R),write(R),
                write('.'),nl,!,fail.
```

2．3　拡張

以上のａ－決定性及びｒ－決定性なる概念は完全に統語論的に与えられてきた。この節の最後に、これらの定義を意味論的に拡張する有力な二つの方法について触れておこう。

(1) r-決定性の定義の条件(3)において、ゴールとその定義体のヘッドとが単一化可能であったとき、そのときの代入情報をそのボディ側のゴールの決定性判定に利用する。すなわち、ゴールp(A)がその定義体のヘッドHiと単一化可能であったとき、そのときのユニフィケーション情報（代入情報）を、Γi かつΔi の決定性判定に利用する。

```
H1 :- Γ1.
 ⋮
Hi :- Γi, [!,] Δi.  (ここで、カット記
        号は存在するなら最右とする)
 ⋮
Hn :- Γn.
```

(2) より一般に、述語の型推論〔例えば、１１〕によって述語引数の型を推論し、ゴールとその定義体ヘッドとの単一化可能性をより詳細に分析する。これはr-決定的なゴールのクラスの拡大を助けるであろう。

以上の拡張法は、一言でいえば、Prologの手続き（述語）間のデータ/ 制御フローの分析を意図した意味的拡張であるといえる。(1)の拡張は簡単であるが、(2)の実現は今後の課題としなければならない。

3．決定性の利用と関連話題

a-決定性とr-決定性はPrologプログラマが日常無意識のうちに用いている直観を自然に定義したものに他ならなかった。それが故に、それらの定義によって捉えられる決定的なゴールのクラスはかなり大きなものと考えられる。ここでは、決定性の意義、利用、及びそれに関連した話題を議論する。

3．1 Prolog プログラムの最適化

我々は論文〔７〕において、述語（呼び出し）の決定性がPrologのような非決定性プログラムを最適化するさいに重要な役割を果たすことを示した。実際、インライン展開、カットの自動挿入、ある種の局所的最適化手法においては述語（呼び出し）の決定性は、Prologプログラムの最適化手法がプログラムに適用可能となるための本質的な条件となっている。そして、純Prologに制限することなく現実のProlog〔１〕を対象としてソースレベルでプログラムの改良のための変換を行うPrologオプティマイザをPrologで試作した〔１２〕、〔１３〕。いくつかの小規模プログラムによる性能評価実験は、典型的なプログラムでは約20% 位の時間効率を達成し、明らかに冗長なプログラムに対しては40～60% の時間効率を達成することを示した。また、最適化手法の一つであるインライン展開は一般にPrologプログラムのような述語（手続き）の並びからなるような言語では、それによるテキストの膨らみが膨大になるように予想されるが、実際には我々の述語（呼び出し）の決定性条件によりプログラムのインライン展開は適度に制限されるため空間効率に悪い影響を与えることはなかった。

一般に、プログラムの最適化の目標、効果はソースからオブジェクトに至る言語階層の異なったレベルでそれぞれ異なっているものである。しかしながら、非決定性のプログラム言語であるPrologの場合では特に述語（呼び出し）の決定性は最適化のあらゆる局面、レベルで効いてくる概念である〔１０〕。また、述語（呼び出し）の決定性は、プログラムの最適化に止まらず、プログラムの知的変換にも重要になってくる概念と考えられる

Debray〔１４〕は我々の決定性の定義を包含する述語の関数性（functionality)を定義し、それに基づいたPrologプログラムの最適化の問題を論じている。述語が関数性をもつとは述語のモードが与えられたとき、その述語の可能な代替節による実行が同じ結果をもたらすことを言う。しかしながら、述語の関数性を一般的に検出することはかなり難しい問題であろう。

3．2　Prologプログラミング方法論

実際のPrologプログラミングでは、プログラムは決定的に書かれるか、もしくは、書いているつもりでいることが多い。また、Prologプログラマはプログラムのデバッグや信頼性の高いプログラムの作成に向けて述語（呼び出し）の決定性をしばしば問題にする〔１５〕。すなわち、述語（呼び出し）の決定性とPrologプログラムの信頼性の間には密接な関係があり、プログラマにとっては述語（呼び出し）の決定性は貴重な信頼性因子と考えられている。逐次型プログラムではプログラムの正当性、停止性、同値性、並行型プログラムではさらにさまざまな究極的な性質(eventuality property)や不変的な性質　(invariant property)等が問題にされるように、

非決定性プログラム言語では述語（呼び出し）の決定性という性質が、特に述語間の呼び出し関係が複雑になればなるほど重要になってくる。

我々の決定性に関する二つのクラスは、信頼性の高いPrologプログラムの作成時に、またプログラムのデバッグのさいにも有力な指針として使うことができる。このように述語（呼び出し）の決定性はプログラムの効率性だけでなく信頼性に大きな寄与を与える。またこの意味では、しばしば論争の的になるPrologのカットは、効率性の面からだけでなく、信頼性の高いPrologプログラムの開発という観点からも議論されるべきである。我々の決定性の定義によって決定的と判定される述語の定義体の中に現れるカットの使われ方は極めて節度のある秩序だった使われ方として参考になるであろう。カットの乱用はPrologプログラムの可読性に大きな影響を与えるが、我々の決定性判定基準はまた適度なカットの使用法のための判断基準の一つとも見ることができる。

3．3　論理プログラミングと制御

述語（呼び出し）の決定性はプログラムの制御構造と密接に関連する問題である。これまで、論理プログラミングのための制御プリミティブにはいくつかの研究が知られている。そのいくつかはPrologの悪名高いカットに代わるものの追求によって動機づけられていたり、論理プログラムを積極的に制御するためのものである。むしろ、論理プログラミングで制御をどう考え、記述するかによって各種のPrologの変種が研究されてきたといってもよい。ここでは述語（呼び出し）の決定性と密接に関係するそのような研究のいくつかを取り上げてみよう。

IC-Prolog〔１６〕は、直接的にそして明示的に実行の決定性を規定する言語構成要素はもたないが、その豊富な制御機構によりPrologプログラムの細かな実行制御を可能にしている。

玉木〔１７〕は論理プログラムの関数サブセットを定義した。そこでは、論理プログラミングの中で入出力の定まった決定的な述語、即ち関数を表現する方法を提案している。例えば、リストを分割するプログラム partition は、

```
function partition(l,x)= l1 ,l2
  partition([x.l],y) = [x. l1 ] ,l2 ← x≦y /
                       l1 ,l2 = partition(l,y).
  partition([x.l],y) = l1 ,  [x. l2 ] ← x>y
                / l1 ,l2 = partition(l,y).
  partition( [] ,y) =  [] ,  [] .
```

と書かれる。

Smolkaは〔１８〕はPrologの素朴なバックトラッキングに基づく制御及びカットに対する反省から、制御を明示的に書くための言語要素をもつSProlog(Self-Documented Prolog)を提案した。そこでは、決定性に関連して、手続きを関数手続きとして宣言したり、手続き呼び出しを関数呼び出しとして表明する方法を提案している。例えば、自然数ｎの階乗ｆを計算する関数手続きfactは

```
fproc fact: ↓integer × ↑integer
```

と書かれ、述語p を関数呼び出しとするときは fc p と書かれる。概念的には、我々のa-決定性の判定は手続きを関数手続きと宣言することに、r-決定性の判定は手続き呼び出しが関数呼び出しであることを表明することに対応する。

Mellish〔１０〕、Bruynooghe〔１９〕は効率的なコンパイリングのために、Prolog述語のモード宣言の利用を提案している。以下では、述語のモード情報が決定性の判定にどのように役立つかを次の例によって見てみよう。

```
:- mode programming-language(+,-).
computer-language(P) :- programming-language(P
                        ,Name),human(Name).
programming-language(lisp,mcCarthy).
programming-language(prolog,kowalski).
programming-langauge(pascal,wirth).
human(mcCarthy).
human(kowalski).
human(wirth).
```

述語呼び出しprogramming-language(P,Name)はモード宣言により(r-)決定的であることがわかる、なぜなら、それの述語定義体のいずれの節も互いに両立しないからである。このように述語のモード宣言は決定性判定に強力な情報を提供する。

Warren〔２０〕は演繹的関係データベースにおける質問文の最適化問題を論じている。Prologプログラムの最適化はこのような質問文の最適化問題とも密接に関係する。Warrenは素朴なバックトラッキングの下では、質問を構成しているゴールの評価順序をどのように並べるのが最も効果的かを検討している。ゴールの評価順序を換える目的は、ゴールの実行の間に考慮されなければならない代替節の数を最小にすることにある。すなわち、非決定性の縮少である。例えば、インスタンシエイトされることが期待されるものでかつ代替節の少ないゴールを先に並べることは非決定性の縮少を助ける。

Nakamura〔21〕は、Debrayと同様の述語の関数性を定義し、その情報をPrologプログラムの実行の制御に用いることを論じている。

4. おわりに

本論文では、a-決定性とr-決定性と呼ばれる二つの決定可能な述語呼び出しの部分クラスを導入し、同値であるがそれらとは異なった定義及びそれらの性質を調べた。これらの概念はPrologのプログラマが日常無意識に用いている直観を自然に、そして統語論的に定義したものであった。さらに、Prologのような非決定性プログラム言語では、述語（呼び出し）の決定性という問題が重要であることを二つの面から論じた。一つはPrologプログラムの最適化という面からであり、もう一つはプログラムの信頼性に関わるプログラムの性質と言う面からである。決定性の、述語のデータ／制御フローアナリシス等に基づく意味的拡張に関する研究は今後の課題である。

謝辞

日頃御指導、御鞭撻を頂く北川敏男会長、及び榎本肇所長に感謝いたします。決定性に関して熱心に議論してくれた竹島卓、横森貴、南俊朗研究員に感謝します。

参考文献

〔1〕Bowen, D. L. : Dec system-10 Prolog user's manual, version 3.43, Dept. of Artificial Intelligence, Univ. of Edinburgh, 1981.

〔2〕Floyd, R. : Non-deterministic algorithms, JACM, Vol. 14, No. 4 (1967), pp. 636-644.

〔3〕Dijkstra, E. W. : A discipline of programming, Prentice-Hall, 1976.

〔4〕Sussman, G. J. : Micro-planner reference manual, MIT AI-Memo 203A, 1971.

〔5〕Shapiro, E. Y. : A subset of concurrent Prolog and its interpreter, ICOT-TR-003, 1983.

〔6〕Kowalski, R. : Logic for problem solving, N-Holland, 1979.

〔7〕Sawamura, H. and Takeshima, T. : Recursive unsolvability of determinacy, solvable cases of determinacy and their applications to Prolog optimization, Proc. of the Symposium on Logic Programming, IEEE Computer Society, Boston, Ma., 1985, pp. 200-207.

〔8〕Byrd, L. : Understanding the control flow of PROLOG programs, Research Paper 151, Dept. of Artificial Intelligence, Univ. of Edinburgh, 1980.

〔9〕Warren, D. H. D. : Implementing Prolog - compiling predicate logic programs, D. A. I Research Report, No. 39 and No. 40, Dept. of Artificial Intelligence, Univ. of Edinburgh, 1977.

〔10〕Mellish, C. S. : Some global optimizations for a Prolog compiler, J. of Logic Programming, Vol. 2, No. 1 (1985), pp. 43-66.

〔11〕Mishra, P. : Towards a theory of types in Prolog, Proc. of the 1984 Int. Symposium on Logic Programming, IEEE Computer Society, Atlantaic City N.J., 1984, pp. 289-298.

〔12〕Sawamura, H., Takeshima, T. and Kato, A. : Source-level optimization techniques for Prolog, IIAS-RR No. 52, 1985, ICOT-TM-0091, also submitted to New Generation Computing.

〔13〕沢村　一：PROLOG プログラムの最適化、国際研和文研究報告　第19号（1986），富士通国際研.

〔14〕Debray, S. K. and Warren, D. S. : Detection and optimization of functional computations in Prolog, LNCS 225, Springer, 1986.

〔15〕近山　隆：Prologプログラミングスタイル、メモ、1983.

〔16〕Clark, K. L. and McCabe, F. G. : The control facilities of IC-Prolog, in Expert Systems in the Micro Electronic Age, edited by P. Michie, Edinburgh Univ. Press, 1979, pp. 122-149.

〔17〕玉木久夫：Prologの関数サブセットP とそれ自身による処理系記述，Proc. of the Logic Programming Conf.,筑波，1983.

〔18〕Smolka, G. : Making control and data flow in logic programs explicit, Proc. of the 1984 Lisp and Functional Programming Language Conf., ACM, 1984, pp. 311-322.

〔19〕Bruynooghe, M. : Adding redundancy to obtain more reliable and more readable Prolog programs, Proc. of the 1st Int. Logic Programming Conf., Marseille, France, 1982, pp. 129-138.

〔20〕Warren, D. H. D. : Efficient processing of interactive relational database queries expressed in logic, VLDB '81, 1984, pp. 272-281.

〔21〕Nakamuara, K : Control of logic program execution based on the functional relation, LNCS 225, Springer, 1986.