

第6章 非決定的な計算

6.1 Prolog 超簡単入門

Prolog は、_____と呼ばれるプログラミング言語の仲間のうち、もっとも代表的なものである。論理型言語では、「～ならば～」という論理式の集まりをプログラムとみなし、論理式の証明をプログラムの実行とみなす。

Prolog などの論理型言語に特徴的な概念としては、_____ (単一化) _____ (後戻り) _____ などが挙げられる。

6.2 Prolog でのプログラミング

Prolog では「～ならば～」という論理式の集まりをプログラムとみなすが、この「～」の部分を作成するのが、次のような形式である。

述語 (項₁, …, 項_n)

このかたちを _____¹ と呼ぶ。「項」は数・文字列などの定数や変数、あるいはリストなどのデータ構造などである。「述語」は直観的には項₁, …, 項_n の間の関係を表す。例えば、father(adam, cain) という素論理式は「Adam は Cain の父である」という関係を表す。

Prolog のプログラムは、_____ (Horn clause) と呼ばれる形式の集まりである。ホーン節とは、素論理式₀ :- 素論理式₁, …, 素論理式_n.

という形である。(最後に必ずピリオド(.)が必要である。):-は左向きの矢印(←)と考えれば良い。つまり、これは素論理式₁, …, 素論理式_n がすべて成り立つならば、素論理式₀ も成り立つという規則を表している。

ホーン節の:-の左側には素論理式は一つのみであり、これを _____ (head) と呼ぶ、:-の右側の素論理式の並びは _____ (body) と呼ぶ。

例えば、

```
grandchild(X, Z) :- child(X, Y), child(Y, Z).
```

は、X と Y, Y と Z の間に child という関係がある (X が Y の子であり、Y が Z の子である) ならば、X と Z の間に grandchild という関係がある (X は Z の孫である) という規則を述べている。ここで X, Y, Z は変数である。Prolog の変数は X, Xs, Y のように、アルファベットの _____ (あるいはアンダースコア“_”)からはじまる識別子 (名前) を使用する。一方、小文字からはじまる識別子は、述

¹アトム (Atom) 複合項などと呼ぶこともある。

語や構成子などの定数に利用される²。変数はどのような項に具体化しても良いので、変数を使用した規則は、実際には無数の規則を表していることになる。

ホーン節のうちボディがないものを _____ (fact) と言う。このときは、:- も省略する。(文末のピリオドは必要である。) 例えば、

```
child(hidetada, ieyasu).
```

は hidetada (秀忠) が ieyasu (家康) の子である、という事実を表明している。ここで hidetada, ieyasu は小文字からはじまるので定数である。

Prolog のプログラムは、このようなホーン節 (事実を含む) を集めたものである。例えば徳川家の家系図を表すプログラムの一部は次のようになる。

```
grandchild(X, Z) :- child(X, Y), child(Y, Z).
```

```
child(hideyasu, ieyasu).
child(hidetada, ieyasu).
child(yoshinao, ieyasu).
child(yorinobu, ieyasu).
child(yorifusa, ieyasu).
```

```
child(iemitsu, hidetada).
child(tadanaga, hidetada).
child(masayuki, hidetada).
```

```
child(ietsuna, iemitsu).
child(tsunayoshi, iemitsu).
```

このようなプログラムをファイルに作成し、処理系にロード³する。例えば Windows 上で動作する SWI-Prolog という処理系ではメニューの「File」—「Consult...」でプログラムファイルをロードすることができる。

プログラムは、このようなホーン節の集まりに対して質問を発することにより起動される。Prolog の処理系は「?- 」というプロンプトを出力するので、このあとに質問を入力する。(質問も最後に必ずピリオドが必要である。)

```
?- grandchild(hidetada, ieyasu).
```

No

```
?- grandchild(iemitsu, ieyasu).
```

Yes

1 番目の質問は、Hidetada (秀忠) は Ieyasu (家康) の孫か? という質問である。これはプログラム中の規則から導出できないので、処理系は No と答えている。2 番目の質問は、Iemitsu (家光) は家康の孫か? という質問である。これは、child(hidetada, ieyasu). と child(iemitsu, hidetada). という二つの事実と

```
grandchild(X, Z) :- child(X, Y), child(Y, Z).
```

というホーン節から導出できるので Yes と答えている。

さらに Prolog では質問の中に変数を含めることができる。すると Prolog の処理系は、_____ を出力する。例えば、

²Haskell と全く逆なので注意する。

³Prolog では伝統的に consult という。

?- grandchild(X, ieyasu).

という質問に対しては、

X = iemitsu

という解を出力する。ここで、リターンキーを押すとこれで終わってしまうが、「;」を入力すると、さらに別解を表示させることができる。

X = iemitsu ;

X = tadanaga ;

X = masayuki ;

No

質問には一般に複数の素論理式を並べることができる。

?- 素論理式₁, ..., 素論理式_n.

このような形を _____ と呼ぶ。直観的には、並べた素論理式がすべて成り立つか?(成り立たせるような変数への代入が存在するか?)という質問を表している。

6.3 単一化(ユニフィケーション)と後戻り

この節では、Prologのプログラムの実行方法を解説する。解説を簡単にするために、まず、最初にいくつかの言葉を定義しておく。

2つ以上の(通常変数を含む)素論理式があり、変数に適切な代入をして、これらの素論理式をまったく同一のものにすることができる時、これらの素論理式は _____ (unifiable) であるという。また、その時の代入を _____ (unifier) という。例えば、次の2つの素論理式

$p(a, Y, Z)$ と $p(X, b, Z)$

は $X = a, Y = b$ を単一化代入として単一化可能である。単一化可能な時は、通常必要なのは最汎(最も一般的な)単一化代入(most general unifier, _____) である。例えば、上の例では、 $X = a, Y = b, Z = c$ という代入も単一化代入であるが、前者の方がより一般的である。実際、この例の場合には $X = a, Y = b$ が MGU になる。

プログラム中のホーン節

$P :- Q_1, Q_2,$
 $\dots, Q_n,$

のヘッド P が素論理式(G)と単一化可能なとき、このホーン節は G に _____ という。

Prologのプログラムの実行方法は、次のようにまとめられる。

1. ゴール節中の _____ 素論理式に対し、適用できるホーン節を選ぶ。適用できるホーン節が複数ある時は、プログラム中に _____ ホーン節から試みる⁴。

⁴Prologを定理証明系として見た時には、ここで、「最も左」、「先に書かれている」という選択をするために、証明系としての力が弱くなってしまふ。つまり、このような制限をしなければ証明できるはずの論理式が証明できなくなってしまう。しかし、プログラミング言語として見た時は、効率を確保するために必要な制限である。

2. 選んだ適用可能なホーン節に対して、その MGU をゴール節の残りの素論理式に適用し、さらにゴール節中の最左素論理式を、上で選んだホーン節のボディに MGU を適用したものと置き換える。(ホーン節のボディがないときは、最左素論理式を消す。)
3. もし適用できるホーン節がなければ、後戻り(_____)して、ひとつの前のゴール節中の素論理式がある状態からやり直す。そして、その素論理式に適用できるホーン節のうち次の候補を選ぶ。
4. ゴール節に素論理式がなくなれば、プログラムの実行を終了し、その時得られた変数への代入を表示する。素論理式がまだあれば、1 に戻る。

具体的に、

?- grandchild(X, ieeyasu).

というゴール節での動作を説明することにする。まずこのゴール節の最左(1つしかないが)素論理式は grandchild(X, ieeyasu) である。これに適用できるホーン節は、今のプログラムでは、

grandchild(X, Z) :- child(X, Y), child(Y, Z).

しかなく、最汎単一化代入は $Z = ieeyasu$ である。するとゴール節は

_____ … ①

に変換される。

次に、この最左の child(X, Y) に適用できる最初のホーン節は、

child(hideeyasu, ieeyasu).

であり、MGU は $X = hideeyasu$, $Y = ieeyasu$ である。すると、ゴール節は、

に変換される。しかし、このゴール節に適用できるホーン節はない。

そこで、上の①のところまでバックトラックし、次の候補である

child(hidetada, ieeyasu).

の適用を試みる。しばらくのあいだ同様にバックトラックが続き、最終的に①のゴール節に対し、

child(iemitsu, hidetada).

というホーン節を選ぶ。そのとき MGU は $X = iemitsu$, $Y = hidetada$ となる。すると、ゴール節は、

に書き換えられる。これはすぐに適用できるホーン節が見つかり、ゴール節が空になって、結果として代入:

$X = iemitsu$

が出力される。ここで;が入力されると、またバックトラックが起こる。

ホーン節の適用は命令型言語・関数型言語の関数呼出しのようなものだと考えることもできるが、単一化により呼び出される側(ホーン節のヘッド)から呼び出す側(ゴール節)に情報が流れることがある、というところが論理型言語に特徴的なところである。例えば、

```
?- child(X, ieyasu).
```

という呼出しに対しては、 $X = \text{hideyasu}$ (あるいは $\text{hidetada, yoshinao} \dots$) という情報が、呼び出された側から、呼び出した側に伝えられる。

Prolog の変数は命令型言語の変数と異なり、いったん単一化により値が代入されれば、以降は値を変更されることはない。ただし、(純)関数型言語の変数とも異なり、最初はいったん不定(unknown)な値を取ることができる。このような振舞いをする変数は一般に _____ (logical variable) と呼ばれる。

6.4 Prologでのリスト処理

Prologでのリストの記法は要素を、`,`で区切り、角括弧(`[]`)で囲んで、`[1, 2, 3]`のように書く。

また、先頭の要素が X で、先頭を除く残りの要素からなるリストが Xs であるようなリストは $[X|Xs]$ と表記される。これはHaskellの _____ という書き方に相当する。`[1, 2, 3]`は実は _____ の略記法である。

Prologでのリストを接続(`append`)するプログラムは次のように記述できる。

```
append([], Y, Y).
append([H|X], Y, [H|Z]) :- append(X, Y, Z).
```

これは、1番目の引数と2番目の引数を接続した結果が3番目の引数になる、という関係を表している。

例えば、`[1, 2]`と`[3, 4]`の接続は次のように求められる。

```
?- append([1, 2], [3, 4], X).
```

```
X = [1, 2, 3, 4] ;
```

```
No
```

問 6.4.1 実際に、上のような結果が出ることを、上に示したPrologの実行方法で1ステップずつ確認せよ。

6.5 UtilNonDet – 非決定性の導入

ここからは Prolog の非決定性を模倣するために、対象言語 Util に非決定性を導入する。非決定性 (nondeterminism) とは _____ を言う。ある選択肢を選んだ結果、計算が失敗する場合がある、その場合は前の選択肢に戻って計算をやり直す (バックトラック)。非決定性は探索型のゲームや構文解析プログラムなどで利用できる。

非決定性の “計算” の型 $L\ e\ a$ は次のように定義する。

```
data L e a = Failure (Maybe e) | Success a (L e a) deriving Show
-- ただし、data Maybe e = Just e | Nothing は Prelude で定義されているデータ型

nil :: L e a
nil = Failure Nothing

unitL :: a -> L e a
unitL a = Success a nil

appendL :: L e a -> L e a -> L e a
(Success x xs) 'appendL' ys = Success x (xs 'appendL' ys)
(Failure _)     'appendL' ys = ys

bindL :: L e a -> (a -> L e b) -> L e b
(Success a es) 'bindL' k = k a 'appendL' (es 'bindL' k)
(Failure e)   'bindL' k = Failure e
```

つまり、 $L\ e\ a$ は基本的には a のリスト型 ($[a]$) だが、空リストに “失敗の理由” として $Maybe\ e$ 型のメッセージが付加されている型である。この L は複数の選択肢を単にリストとして表現している。

$unitL$ と $bindL$ は、リストの内包表記を説明する時に使った $unit :: a \rightarrow [a]$ と $bind :: [a] \rightarrow (a \rightarrow [b]) \rightarrow [b]$ に類似の関数である。

計算の失敗は空リストに対応する $Failure$ で表される。

```
failL :: e -> L e a
failL e = _____
```

$UtilNonDet$ は、構文は $UtilErr$ と同じである。つまり、以下の構文を持つ。

$$Expr \rightarrow \dots \mid \text{try } Expr \text{ catch } Expr$$

$\text{try } m \text{ catch } h$ は、 m を評価し、成功した場合は、 h に $Nothing$ を渡す。 m の評価に失敗した場合は、“失敗の理由” $Just\ e$ を h に渡す。この h は “バックトラック” が起こったときに評価される。

tryL は次のように定義される。

```
tryL :: L e a -> (Maybe e -> L e a) -> L e a
tryL (Success v vs) h = Success v (tryL vs h)
tryL (Failure e) h     = h e
```

“失敗の理由” e を第 2 引数に渡していることを除けば、単なるリストの接続と考えることができる。

ここで、 $L\ e\ a$ から通常のリスト $[a]$ への変換関数 toList を次のように定義しておく。

```
toList :: L e a -> [a]
toList (Success x xs) = x : toList xs
toList _              = []
```

UtilNonDet プログラム

```
test0 = (try 1 catch \ _ -> 2) * (try 3 catch \ _ -> 4)
```

をコンパイルすると、次の Haskell プログラムが得られる。

```
test0 = tryL (unitL 1) (\ _ -> unitL 2) 'bindL' \ x ->
        tryL (unitL 3) (\ _ -> unitL 4) 'bindL' \ y ->
        unitL (x*y)
```

この、test0 は [x*y | x <- [1,2], y <- [3,4]] というリスト内包表記に対応する。toList test0 は、[3,4,6,8] となる。

また、次の UtilNonDet プログラム

```
test1 = (try 1 catch \ _ -> 2) / (try 0 catch \ _ -> 4)
```

をコンパイルすると、次の Haskell プログラムが得られる。

```
test1 = tryL (unitL 1) (\ _ -> unitL 2) 'bindL' \ x ->
        tryL (unitL 0) (\ _ -> unitL 4) 'bindL' \ y ->
        if y == 0 then failL "Division by 0" else unitL (x/y)
```

toList test1 は、[0.25,0.5] となる。失敗している計算については結果に現れていないことに注意する。同じプログラムを UtilErr でコンパイルすると、UtilErr ではバックトラッキングが起これないので、この計算は全体が失敗に終わる。

なお、次の headL を用いてリストの頭部を取ることににより、成功した最初の計算だけを返すことも可能である。

```
headL :: List e a -> a
headL (Success x _) = x
```

headL test1 の値は 0.25 となる。この場合、Haskell が _____ を採用しているため、他の選択肢の計算は行なわれない。そのため選択肢が無数あるような場合でも最初の選択肢の計算結果を出力することができる。

さらに Prolog にはカットといって後戻りの振舞いを変更する機構があるが、このような機構を導入するためには、後の章で紹介する接続 (continuation) のような仕組みが必要である。

問 6.5.1 非決定性と状態の両方の特徴を持つ計算の型として、

```
type STL s a = s -> ([a], s)
type LST s a = s -> [(a, s)]
```

の2つのバリエーションが考えられる。このそれぞれに対して、インタプリタの定義を完成させ、2つの違いを説明せよ。

この章の参考文献

- [1] Ralf Hinze 「Prological Features in a Functional Setting Axioms and Implementations」
Third Fuji International Symposium on Functional and Logic Programming, 1998 年
Prolog のカット等のオペレータの意味を整理している。