プログラミング言語論

論理型プログラミング言語

水野嘉明

目次

2

1. 論理プログラミング

3

復習

1.1 論理型言語

- 公理(論理式)の集合を定義し、推論規則の適用により計算を行う
  - *単一化*(unification)が基本操作
- Prolog が代表
  - 1972 カルメラウア、コワルスキー
    (実用的な論理型言語は、現在Prologのみ)

4

復習

1.1 論理型言語

- Prologの応用
  - 自然言語処理
  - アルゴリズムの記述
  - データベースの探索
  - コンパイラの記述
  - エキスパートシステムの構築

※ パターン照合、後戻り検索、不完全な情報を扱う応用に向いている

5

復習

1.2 論理プログラミング

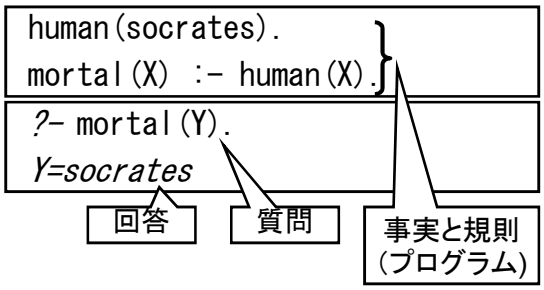
- 論理プログラミングとは
  (1) 情報の表現のための事実と規則の使用
  (2) 質問の応答への論理推論の使用
  - プログラマが事実と規則を記述し、質問を与える。処理系が論理推論を用いて質問への回答を計算する

6

復習

1.2 論理プログラミング

- Prologの例



7

1.2 論理プログラミング

- 論理プログラミングでは、
  関数ではなく、*関係*(relation)を扱う
  - 引数と結果を同様に扱うので、関数プログラミングよりも柔軟である

8

1.3 述語

- *述語*(predicate)
  - 何らかの関係や事実を、述べる語
  - 関数のように、引数をとることができる
  - 真か偽に評価される
    (関数のように数値等の値を返すわけではない)

9

1.3 述語

- 新たな述語を論理式で定義することによってプログラムを作り上げていく
- 基本構造

  述語名(項, 項, ・・・, 項)

10

1.3 述語

- 例

「taro は hanako の父親である」

```
father(taro, hanako).
```

事実を述べておく

システムは、様々な質問に答えることが出来る

11

1.3 述語

質問とその回答 (1)

```
?- father(taro, hanako).
true.
```

質問(述語)が、真か偽かを答える

12

1.3 述語

質問とその回答（2）

```
?- father(taro, X).
X = hanako .
?- father(Y, hanako).
Y = taro .
```

X、Yは変数。質問を真にするような変数の値を求めて、答える。

13

1.3 述語

- Prologの述語について
  - 慣例として、述語名／引数の数、でPrologの述語を表現する
    （例） atom/1
  - 述語名が同じであっても、引数の数が異なれば、別の述語として扱う
  - Prologの処理系には、組込み述語が用意されている

14

2. Prolog入門

2.1 基本構文
2.2 プログラム
2.3 質問
2.4 否定
2.5 演算

15

2.1 Prolog入門：基本構文

- Prolog の基本構文

事実

規則

質問

<fact> ::= <term> .
<rule> ::= <term> :- <terms> .
<query> ::= <terms> .
<term> ::= <number> | <atom> |
<variable> | <atom>(<terms>)
<terms> ::= <term> | <term>,<terms>

16

2.1 Prolog入門：基本構文

- *項*(term)
  - ➢事実(fact)、規則(rule)、質問(query)は、項(term)を用いて指定される

17

2.1 Prolog入門：基本構文

- ➢単項 (single term) とは
  - ●数 (number)
  - ●アトム (atom)
    小文字で始まる、それ自身を表す
  - ●変数 (variable)
    大文字で始まる

[例]
0 19.72 lisp algol60 X Source

18

2.1 Prolog入門：基本構文

- 複合項（compound term）とは
  - アトムと括弧でくくられた部分項の並び

```
link(bcpl, c)
```

述語は複合項により表記される

19

2.1 Prolog入門：基本構文

- 複合項に関する注意
  - 若干の演算子は、前置記法だけではなく、中置記法で書かれることもある

[例]

=(X, Y) ⇒ X=Y

20

2.2 Prolog入門：プログラム

- Prologのプログラム
  - 事実、規則の集まりをファイルにしておく
  - 組込み述語 consult/1 にて読込む

述語名　引数の数

21

2.2 Prolog入門 :プログラム

- 規則(rule)と事実(fact)
  - 規則の一般形

$head :- body_1, body_2, \cdots, body_N.$

$body_1$ から $body_N$ までがすべて成り立てば head が成立する、ということを表す

22

2.2 Prolog入門 :プログラム

- 規則は、含意(implication)

$body_1 \wedge \cdots \wedge body_N \rightarrow head$

を表している

ホーン節で書くと

$head \vee \overline{body_1} \vee \overline{body_2} \vee \cdots \vee \overline{body_N}$

23

2.2 Prolog入門 :プログラム

- head を頭部(ヘッド部)、残りのbodyをまとめて体部(ボディ部)と呼ぶ

```
path(L,M) :- link(L,X), path(X,M).
```

ヘッド部　　ボディ部

24

2.2 Prolog入門 :プログラム

- 事実(fact)は、
  ヘッド部のみで、ボディ部のない
  特別な規則
  とみなす事ができる

```
link(fortran, algol60).
```

25

2.2 Prolog入門 :プログラム

- プログラム例

```
% links of programming languages

link(fortran, algol60).
link(algol60, cpl).
link(cpl,     bcpl).
link(bcpl,    c).
link(c,       cplusplus).
```

（続く）

26

2.2 Prolog入門 :プログラム

（続き）

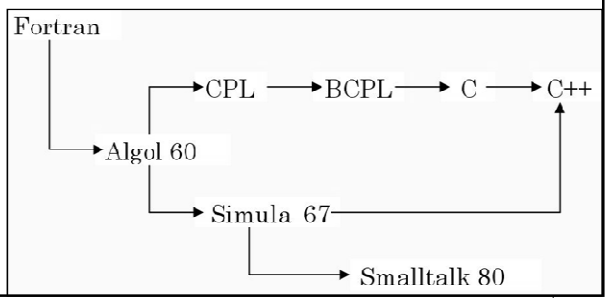
```
link(algol60,  simula67).
link(simula67, cplusplus).
link(simula67, smalltalk80).

path(L,L).
path(L,M) :- link(L,X), path(X,M).
```

27

2.2 Prolog入門 :プログラム

このプログラムは、次のようなプログラミング言語の関係を表している



2.2 Prolog入門 :プログラム

- 次の事実

```
path(L,L).
```

は、
すべてのLについて
path(L,L) が成り立つ
と読むことが出来る

29

2.2 Prolog入門 :プログラム

次の規則の意味は、

```
path(L,M) :- link(L,X), path(X,M).
```

すべての L, Mについて、
もし、あるXが存在して
link(L,X) かつ path(X,M) ならば
path(L, M) が成り立つ

30

2.2 Prolog入門 :プログラム

- 結局このプログラムは、図の
  - 「矢印」を ″link″ で表し、
  - 「矢印をたどってたどり着く道があること」を ″path″ で表している

31

2.2 Prolog入門 :プログラム

- コンサルト(consult) は、事実と規則からなるプログラムファイルを読込み、その内容を「規則データベース」に追加する

32

2.2 Prolog入門 :プログラム

- プログラムの読込み

```
?- consult( 'links.swi' ).
% links.swi compiled 0.00 sec, 1,652 bytes
true.
?-
```

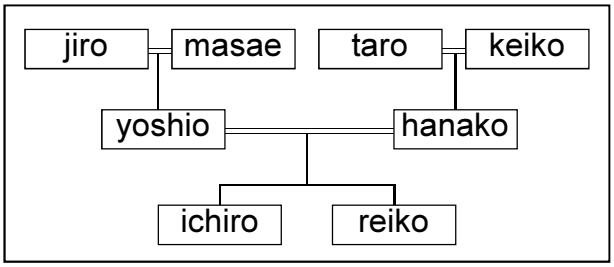
33

2.2 Prolog入門 :プログラム

- 演習 8.1
  - ➢次ページの親子関係を、Prologプログラムにて記述せよ
  - ➢次のような述語を作ること
    - father
    - mother
    - parent　（fatherとmotherで定義）
    - ancestor （parentで定義）

34

2.2 Prolog入門 :プログラム



35

2.3 Prolog入門 : 質問

- 質問（query）
  - ➢Prologの実行は、Prologの処理系に対し、質問となる述語を与えることによる
  - ➢プロンプト「 ?- 」の後に、質問を入力する

36

2.3 Prolog入門：質問

- この実行時に入力する述語列（質問となる述語列）を、*ゴール節*と呼ぶ

（処理系は、ゴール到達を目指して推論を進める）

37

2.3 Prolog入門：質問

- 次の質問

$\langle term \rangle_1, \langle term \rangle_2, \cdots, \langle term \rangle_k .$

の意味は、

$\langle term \rangle_1$ かつ $\langle term \rangle_2$ かつ ・・
かつ $\langle term \rangle_k$ であるか？

38

2.3 Prolog入門：質問

- 変数のない質問

```
?- father(taro,hanako),
        father(hanako,ichiro).

   false.
```

応答は、単に true(yes) か false(no) である

39

2.3 Prolog入門：質問

- 質問文中の変数は、適当な対象が存在するか否かを問うている

```
?- father(jiro,L),father(L,M).
```

これは、次のような意味

father(jiro,L) かつ father(L,M) となるような L, Mが存在するか？
存在するならば、その値は？

40

2.3 Prolog入門：質問

- *解*(solution)
  - 質問への解とは、
    質問文(ゴール節)を真とするような、変数の値への束縛
    である
  - 解を持つ質問文を、
    充足可能(satisfiable)である
    という

41

2.3 Prolog入門：質問

- 充足可能な質問に対するシステムの応答

```
?- father(jiro,L),father(L,M).
 L = yoshio,
 M = ichiro
```

別解がある時は、システムはユーザの指示待ちとなる

42

2.3 Prolog入門：質問

◆セミコロンを入力
　次の解を表示する
◆改行(Enter)を入力
　解の表示を終了し、次の質問を
　受け付けるプロンプトを表示する

（注）細かな動作は、システムにより
　　異なることがある

43

2.3 Prolog入門：質問

セミコロンを入力した例

```
?- father(jiro,L),father(L,M).
L = yoshio,
M = ichiro ;
L = yoshio,
M = reiko.
```

';'を入力

44

2.4 Prolog入門：否定

- 失敗による否定（negation as failure）
  - ➢質問文を充足するのに失敗
  - ⇒ Prologは false（no）と応える

証明することが出来なければ、
偽に違いない

45

2.4 Prolog入門：否定

【例】前述のプログラムは、tarooの親については何も言っていない

```
?- father(saburo, taro).
   false.
```

46

2.4 Prolog入門：否定

- 否定演算子 not
  - 質問 not(P) は、システムが Pを導き出せなければ真として扱われる

注：notは、複雑な文の場合は使用方法に注意が必要

47

2.4 Prolog入門：否定

- not演算子の使用例

```
?- parent(L,N), parent(M,N), not(L=M).
L = jiro,
N = yoshio,
M = masae ;
L = taro,
N = hanako,
M = keiko ;
   :
```

48

2.4 Prolog入門：否定

前ページの例と比較

```
?- not(L=M), parent(L,N), parent(M,N).
   false.
```

質問が始まった時点で、L、Mの値は不明である。不明な値は等しくなりえるので、not(L=M)は失敗する。

49

2.5 Prolog入門：演算

- 四則演算
  - ＋，－，＊，／，／／，mod
    / は実数の除算、// は整数の除算、mod は剰余
- 比較演算（数値の比較）
  - ＜，＞，＝＜，＞＝，＝:＝，＝￥＝

50

2.5 Prolog入門：演算

- 演算結果を変数にセットするには is を使用する

```
?-  X is 1 + 2 + 3.
 X = 6.
```

is の右辺の式を評価し、左辺の変数にセットする

51

2.5 Prolog入門：演算

注：is は、通常の代入とは異なる

```
?- X is 3, X is 4.
   false.
```

最初のisでは、パターンマッチング（単一化という＝後述）により、変数Xは3となる。次のisでは、Xは3なのでマッチングに失敗する。

52

2.5 Prolog入門：演算

- 演算の例（1）

2乗を求めるプログラム

```
square(X, Y) :- Y is X * X.
```

実行結果

```
?- square(5, Y).
   Y = 25.
```

53

2.5 Prolog入門：演算

- 演算の例（2）

階乗を求めるプログラム

```
% 階乗を求める
fact(0, 1).
fact(N, F) :- N > 0, N1 is N - 1,
              fact(N1, F1),
              F is N * F1.
```

54

2.5 Prolog入門：演算

- 注:この例は、以下では不可

```
% 階乗(不可な例)
fact(0, 1).
fact(N, F) :-  N > 0,
               fact(N - 1, F1),
               F is N * F1.
```

「N-1」が、評価(計算)されない

55

2.5 Prolog入門：演算

- 演習8.2
  - フィボナッチ数を求めるプログラム fibを、作成せよ
  - ★ フィボナッチ数の定義

    fib(0) = 0, fib(1) = 1

    fib(n) = fib(n−1) + fib(n−2)

    (ただし、n≧2)

56

3. Prologの動作

3.1　単一化

3.2　バックトラック

3.3　カット

57

3.1 Prologの動作：単一化

- *具体化*
  - 項の中の変数を、具体的な値により置換すること
  - 同じ変数は、同じ値で置換しなければならない
- *具体値*（instance）
  - 具体化の結果

58

3.1 Prologの動作：単一化

- 例えば

```
?- f(X, b) = f(a, Y).
 X = a,
 Y = b.
```

f(a, b)は、f(X, b)および f(a, Y)の具体値である

59

3.1 Prologの動作：単一化

- *単一化*（unification）
  - 二つの項T_1とT_2が同一になるような変数の具体化を見つけ適用する（変数が両方の項にあるときは、同じ値で具体化する）
  - 二つの項が共通の具体値を持つ時に、単一化可能

60

3.1 Prologの動作 : 単一化

- 関係 identity が、次の事実により定義されているとする

```
identity(Z,Z).
```

次の応答を計算するため、単一化が使用される

```
?- identity( f(X,b), f(a,Y) ).
 X = a,
 Y = b.
```

61

3.1 Prologの動作 : 単一化

次の二つを単一化

identity(Z,Z)

⇕

identity(f(X,b),f(a,Y))

これにより、以下も単一化される

f(X,b) ⇔ Z ⇔ f(a,Y)

62

3.2 Prologの動作:バックトラック

- Prologプログラムの動作の流れ
  1. ゴール節の述語と一致する述語をヘッド部にもつ節を探す
  2. その節のボディ部の各述語が真になるかを調べる
  3. ボディ部の述語がすべて真になれば、trueを返して終了する

63

3.2 Prologの動作：バックトラック

4. 別の節のヘッド部で一致するものを探す
5. みつかったら、2.へ。見つからなかったらfalseを返して終了

この動作を、*バックトラック*という

64

3.2 Prologの動作：バックトラック

- バックトラックの例
  - 次の質問を考える

```
?- parent(keiko, hanako).
```

どのような順序で、*true.* を返すのか？

65

3.2 Prologの動作：バックトラック

① ヘッド部に述語 parent を持つ節を探す
⇒ parent(X,Y) :- father(X,Y). を見つける

② {X=keiko, Y=hanako} で単一化
⇒ father(keiko, hanako) を真とできれば成功

66

3.2 Prologの動作：バックトラック

③ ヘッド部に father を持つ節を探す
⇒ father(jiro, yoshio). が見つかるが、単一化できず失敗
⇒ バックトラック

④ 別の father をヘッド部とする述語を探す
⇒ 同様に単一化できず、失敗（繰り返す）

67

3.2 Prologの動作：バックトラック

⑤ バックトラックして、別の節でヘッド部に述語 parent を持つものを探す
⇒ parent(X,Y) :- mother(X,Y). を見つける

⑥ {X=keiko, Y=hanako} で単一化
⇒ mather(keiko, hanako) を真とできれば成功

68

3.2 Prologの動作：バックトラック

⑦ ヘッド部に mother を持つ節を探す
⇒ mother(masae, yoshio). が見つかるが、単一化できず失敗
⇒ バックトラック

69

3.2 Prologの動作：バックトラック

⑧ 別の mother をヘッド部とする述語を探す
⇒ mother(keiko, hanako). を見つける
⇒ mother(keiko, hanako) が真
⇒ parent(keiko, hanako) が真
⑨ ゴール節が真となったので、true. を返して終了

70

3.2 Prologの動作：バックトラック

- 強制バックトラック
  - trueとなった（解が一つ見つかった）後も、ユーザが セミコロンを入力すると、バックトラックを行い次の解を探す

71

3.2 Prologの動作：バックトラック

- 強制バックトラックの例

```
?- father(jiro,L),father(L,M).
L = yoshio,
M = ichiro ; ⇐
L = yoshio,
M = reiko.
```

72

3.3 Prologの動作 : カット

- バックトラックを続けると、探索空間全体を調べる
  - 余分な(無駄な)探索を続けることがある

カット により、バックトラックを制御する

73

3.3 Prologの動作 : カット

- 組込み述語 ！を評価(実行)すると、以降の節を探索しない
  (そこを後戻りできない)
  - カットは、最初の実行では無条件に成功する
  - それ以前に実行されたゴールの再試行と、次の規則の探索を禁止

74

3.3 Prologの動作 : カット

- カットの動作 (1)

head :- $goal_A$, ..., $goal_M$, !, $goal_N$, ..., $goal_Z$.

再試行は行われない　　再試行する

一方通行

75

3.3 Prologの動作 : カット

➢カットの動作 (2)

$head_1$:- $goal_A$, !, $goal_B$.
$head_2$:- $goal_C$, $goal_D$.
:
$head_{13}$:- $goal_Y$, $goal_Z$.

カットは $goal_A$ の再試行だけではなく、次候補節の選択もカットする

76

3.3 Prologの動作 : カット

● カット利用の例 ＝ 条件判断

```
condSubr(X) :- X =:= 1, !, print('One').
condSubr(X)  :-  print('not One').
```

xが1でなければ、次の規則を調べる

カット(!) がないと、1の時もバックトラックにより 'not One' が表示されてしまう

77

【参考】Prolog 参考URL

- SWI-Prolog's home
  http://www.swi-prolog.org/
- お気楽 Prolog プログラミング入門
  http://www.geocities.jp/m_hiroi/prolog/
- Prolog入門
  http://bach.istc.kobe-u.ac.jp/prolog/intro/

78

お疲れさまでした