関数・論理型プログラミング実験
Prolog演習第1回
（通算第11回）

松田 一孝

TA: 武田広太郎 寺尾拓

前回の発展課題の補足

- 構文解析コンビネータは
  効率を追求するとモナドとすることを
  諦める必要がある場合があるかも？
  - そのとき，その理由を明記してくれた
    諦めてもOK
  - というか，構文解析コンビネータは
    ApplicativeやArrowのkiller example
    の一つ

今日からProlog演習

- 予定
  - 第1回
    - Prologの使い方
  - 第2回
    - Prologの評価メカニズム
  - 第3回
    - 探索について（仮）
  - 第4回
    - 関数論理型言語Curry

Prolog？

- 論理プログラミング言語
  - 一階述語論理に基づく
    宣言的プログラミング記述

- 非決定的計算 + built-in search
  - generate-and-testプログラミング

どうしてProlog？

- 新たな問題解決の視点
  - 「ハンマーを持つ人には、すべてが釘に見える」にならないように
    - 道具をたくさん持つのは，問題をより自然でエレガントに解くことに通じる
  - 非決定的計算 + built-in search の強力さ（および落とし穴）
    - HaskellならMonadで，Curryなら無償で利用できる

まずは使ってみよう

family.pl

```
/* family.pl */
male(kobo).
male(koji).
male(iwao).
female(sanae).
female(mine).
```

koboはmaleである
というfactの宣言

起動，読み込み

```
$ swipl
...
...
?- ['family.pl'].
% family.pl compiled 0.00 sec, 7 clauses
true.

?-
```

問い合わせ

```
?- male(kobo).
true.

?- female(sanae).
true.

?- female(koji).
false.
```

問い合わせ，続

maleであるようなXは？

```
?- male(X).
X = kobo ;
X = koji ;
X = iwao.

?- male(X).
X = kobo .
```

「;」他にある？
(tabキーでも可)

「.」もういいや
(Enterキーでも可)

```
/* family.pl */
male(kobo).
male(koji).
male(iwao).
female(sanae).
female(mine).
```

親子関係

```
/* family.pl */
male(kobo).
male(koji).
male(iwao).
female(sanae).
female(mine).
parent(kobo, koji).
parent(kobo, sanae).
parent(sanae, iwao).
parent(sanae, mine).
```

「koboとkojiは
parentという関係である」
というfactの宣言

使ってみる

```
?- ['family.pl'].
% family.pl compiled 0.00 sec, 2 clauses
true.

?- parent(kobo,X).
X = koji ;
X = sanae.

?- parent(X,iwao).
X = sanae.
```

父親，母親

「XとYがfatherなのは，
XとYがparentで，Yがmaleのとき」
というruleの宣言

```
/* family.pl */
…略…
father(X, Y)  :- parent(X, Y), male(Y).
mother(X, Y)  :- parent(X, Y), female(Y).
```

←

問い合わせ例

```
?- mother(kobo,X).
X = sanae.

?- father(kobo,X)
X = koji ;
false.

?- father(_,X).
X = koji ;
X = iwao ;
false.

?- mother(_,X).
X = sanae ;
X = mine.
```

注：prologはこの時点ではsanaeがfatherでないことを知らない

「_」はワイルドカード

祖父母，祖先

```
/* family.pl */
…略…
grandparent(X, Z) :- parent(X, Y), parent(Y, Z).

ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
```

問い合わせ例

```
?- grandparent(kobo,X).
X = iwao ;
X = mine.

?- ancestor(kobo,X).
X = koji ;
X = sanae ;
X = iwao ;
X = mine ;
false.

?- grandparent(kobo,X),male(X).
X = iwao ;
false.
```

注意：述語やruleの順番

最初にparent(X, Y)かどうか確認

```
ancestor(X, Y)  :- parent(X, Y).
ancestor(X, Y)  :- parent(X, Z), ancestor(Z, Y).
```

次に，parent(X, Z)であるようなZに対し，
ancestor(Z, Y)となるか確認

なので，次は動かない

XとYがancestorか確認するには…
まず，ZとYがancestorな関係であるZについて…

```
ancestor(X, Y)  :- ancestor(Z, Y), parent(X, Z).
ancestor(X, Y)  :- parent(X, Y).
```

否定

```
?- grandparent(kobo,X),male(X).
X = iwao ;
false.

?- grandparent(kobo,X),\+ male(X).
X = mine.
```

否定：Prologは，male(X)だと証明できなかったとき\+ male(X)であるとする
(negation as failure)

注意：Negation as Failure

- 通常の述語論理
  - 事実の追加で，
    成り立たなくなる式はない
- Prolog
  - 事実の追加で，
    成り立たなくなる式がある

```
?- grandparent(kobo,X),\+ male(X).
false.
```

```
/* family.pl */
…略…
male(mine).
```

注意，続

- 次は予期せぬ振舞いをする

```
mother_(X, Y)  :- \+ male(Y), parent(X, Y).
```

```
?- mother_(kobo, sanae).
true.

?- mother_(kobo, X).
false.

?- \+ male(X).
false.
```

male(X)が成功するため否定は失敗

これまでのまとめ

- .plにfactやruleを書く
- インタプリタで問い合わせができる
  - father(kobo, X). のような問い合わせを行うとPrologがXを計算してくれる

大雑把な構文

コメント

- C風
  - /* comment */
  - ただしネストできる

項

- 定数
  - kobo, koji, sanae, …
  - 整数
  - 文字列 'ma oyari'
- 変数
  - X, Y, Ab, _C, …
- 複合項
  - f(複合項, …, 複合項)
    - fをfunctorと呼ぶ
    - fのsyntaxは整数でない定数と同じ
  - s(z), s(s(z)), など

fact

- 述語(項1, …, 項n).

```
/* family.pl */
male(kobo).
male(koji).
male(iwao).
female(sanae).
female(mine).
parent(kobo, koji).
parent(kobo, sanae).
parent(sanae, iwao).
parent(sanae, mine).
```

rule

- 述語(項1, …, 項n) :-
  述語1(項11, . . . , 項1n_1), …,
  述語m(項m1, . . . , 項mn_m).

```
/* family.pl */
…略…
grandparent(X, Z) :- parent(X, Y), parent(Y, Z).

ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
```

リスト記法

- $[t_1|t_2]$
  - コンス
- []
  - 空リスト
- $[t_1, t_2, t_3]$
  - $[t_1 | [t_2 | [t_3 | []]]]$のこと
- $[t_1, t_2 | t_3]$
  - $[t_1 | [t_2 | t_3]]$のこと

- Prologは型なしなので[[1],2]も可

もう少し「プログラムらしい」例

加算（a la Peano）

```
add(z, Y, Y).
add(s(X), Y, s(Z)) :- add(X, Y, Z)
```

```
?- add(s(z), s(z), Z).
Z = s(s(z)).

?- add(s(z), s(s(z)), Z).
Z = s(s(s(z))).

?- add(X, s(z), s(s(z))).
X = s(z) ;
false.
```

減算も
（この定義では）

注意

- どれを変数にして
  問い合わせできるかは定義による

```
addA(s(X), Y, Z) :- addA(X, s(Y), Z).
addA(z, Y, Y).
```

```
?- addA(s(z), s(z), Z).
Z = s(s(z)).

?- addA(X, s(z), s(s(z))).
ERROR: Out of local stack
...
```

連接

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

```
?- append([1, 2, 3], [4, 5], Z).
Z = [1, 2, 3, 4, 5].

?- append([_, _], Y, [1, 2, 3, 4, 5]).
Y = [3,  4,  5].
```

第11回レポート課題
締切 7/8 13:00

問1

- XとYが兄弟姉妹の関係にあることを表現する述語sibling(X, Y)を定義せよ
  - sibling(X, X)は偽になるようにせよ
- 適当な家族の親子関係をmale, female, parent述語を用いて定義しsiblingが動作することを確認せよ

問2

- XとYが血縁関係にあることを表す二項述語 bloodrelativeを定義せよ
  - 適当な例について動作を確認せよ
    - 冒頭の例だと
      - sanaeとkoji，kojiとiwaoは血縁ではない
      - koboはkoji，sanae，iwao，mineと血縁関係にある
  - 注意
    - 血縁：共通の祖先を持つ
      - なので，いとこも血縁

問3

- 以下の述語を実装せよ
  - XとYが逆順になっていることを表す述語reverse(X, Y)
    - ただし以下となるように
      - ?- reverse ([1, 2, 3], X).
        X = [3, 2, 1]
  - Xの各リストを順に連接したものがYであることを表す述語concat(X, Y)
    - ただし以下となるように
      - ?- concat([[1], [2, 3]], X).
        X = [1, 2, 3]

問4

- Vを頂点のリスト，Eを枝のリストとしたときハミルトン経路を持つかどうかを判定する述語hamilton(V,E)を書け

ヒント

- 以下を実装する？
  - Pがグラフ(V,E)のシンプルなパスかを判定する述語 simple_path(P,V,E)
    - シンプルなパス：同じノードを二つ含まないパス
  - Yの全要素がXの要素であることを判定する述語contains(X,Y)

  - hamilton(V,E)
    :- simple_path(P,V,E),contains(P,V)

発展

- 本日説明した範囲の構文（w/o \+）でPrologがTuring完全であることを示せ