プログラミング言語論

第10回

Prolog入門

担当：犬塚

今日の講義

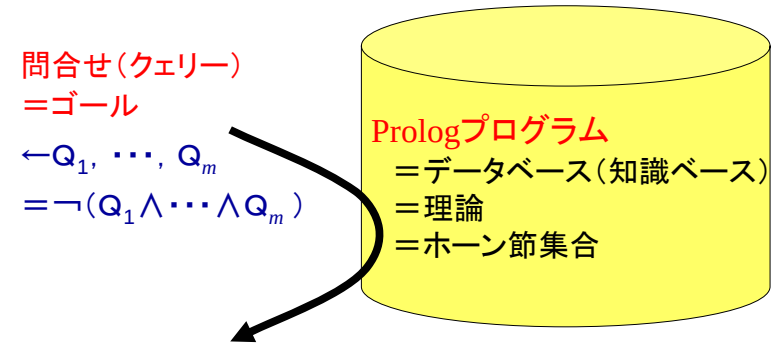
論理型の代表的プログラミング言語Prologを概観する

- Prologの構文とプログラミング
- Prologプログラムの実行
- バックトラック
- 実行の制御：カット
- データベース操作

節形式（clausal form）

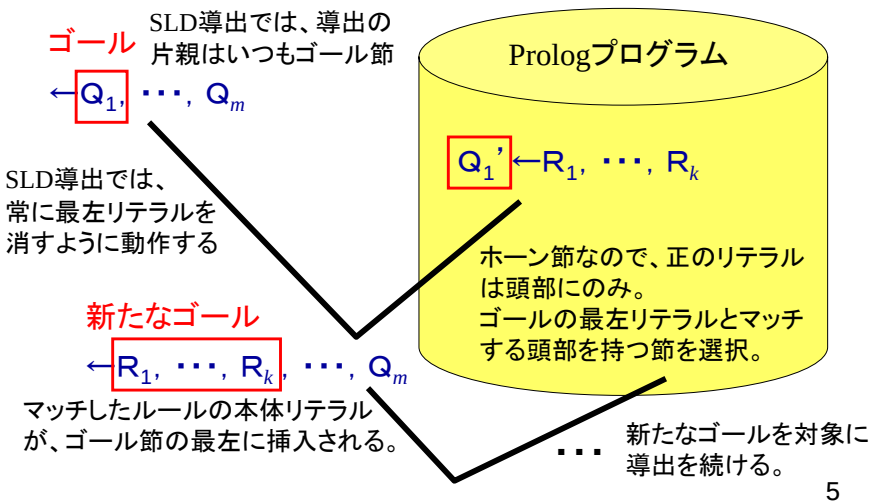
- 節形式：
  $P_1 \vee \cdots \vee P_n \leftarrow Q_1 \wedge \cdots \wedge Q_m$
  $= P_1, \cdots, P_n \leftarrow Q_1, \cdots, Q_m$
- n≦1のときホーン節（Horn clause）という。
  - $P \leftarrow Q_1, \cdots, Q_m$ ：規則（ルール）節
  - $P \leftarrow$ ：事実（ファクト）節
  - $\leftarrow Q_1, \cdots, Q_m$ ：目的（ゴール）節
  - $\leftarrow$ ：矛盾節（空節）
- $P \leftarrow Q_1, \cdots, Q_m$のとき、
  - Pを頭部（head）、$Q_1, \cdots, Q_m$を本体(body)という。

Prologの動作環境



YES（問合せがDBと矛盾する）　＝成功
　または
NO（問合せがDBと矛盾しない）　＝失敗
＋YESとする置換 θ

SLD導出



Prologの構文

- Prologはすべて節の形式でプログラムを書く。
- 4つの形式に対応して、

ルール節

P :− Q_1, ・・・, Q_m.

ファクト

P.

ゴール

？−Q_1, ・・・, Q_m.（プログラムの実行、問合せの場合）

：−Q_1, ・・・, Q_m.（プログラム中に書く場合）

空節

特になし。（プログラムとして書かない）

例

- 事実
  - 次郎は太郎の持っているものをなんでも欲しがる。
  - 太郎は本を持っている。
- 質問
  - 次郎は本を欲しがるか？

has(x,y)を　「xはyを持っている」を表す述語、
wants(x,y)を「xはyを欲しがる」を表す述語、
taro, jiro, bookを太郎、次郎、本を表す定数とすると、

```
wants(jiro,Y):−has(taro,Y).
has(taro,book).
？−wants(jiro, book).
```


Prologの構文（つづき）

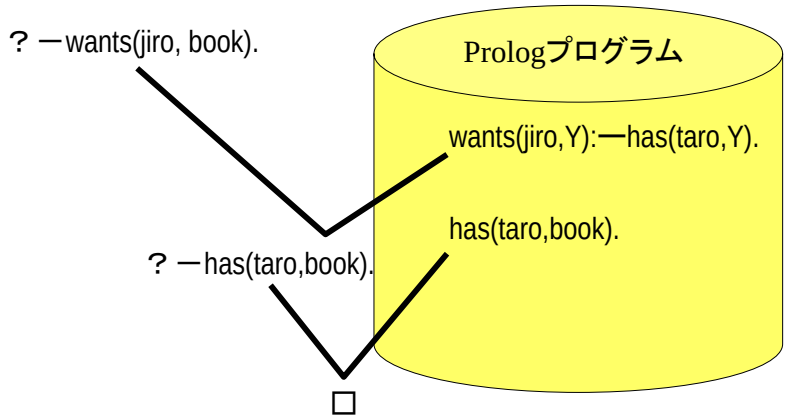
```
wants(jiro,Y):−has(taro,Y).
has(taro,book).
？−wants(jiro, book).
```

- リテラルは述語記号と項（変数、定数）からなる。
- 述語記号、定数、変数は宣言なしで使ってよい。
- 述語記号、定数記号は、小文字のアルファベットではじめる。
- 変数記号は大文字のアルファベットではじめる。

- これ以外に、関数記号を用いても良い。

```
wants(brother(X),Y):−has(X,Y).
has(taro,book).
？−wants(brother(taro), book).
```

動作例



練習

質問に答えられるようにProlog形式で節を与えよ。

- 洋子(yoko)はセーター(sweater)を買った(bought)。

質問

- 洋子は洋服(clothes)を持っている(has)か？

暗黙的に分かっていること:

- セーターは洋服である(isa)。
- AがBを買えば、AはBを持っている。
- AがBをもっており、AがCであれば、AはCを持っている。

変数の解釈

- 節に含まれる変数は全称限定されている。
- 本体リテラルは負のリテラルであるので、そこに含まれる変数は存在限定されていると解釈される。

$\forall X \forall Y\ p(X) \leftarrow q(X,Y)$

$= \forall X \forall Y (p(X) \vee \neg q(X,Y))$

$= \forall X\ p(X) \vee \neg (\exists Y\ q(X,Y))$

$= \forall X\ p(X) \leftarrow \exists Y\ q(X,Y)$

例

次のルールは、祖父の述語の定義とみなせる。

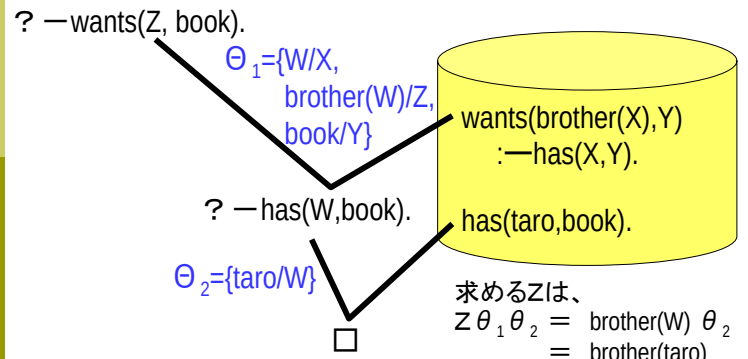
grandFather(X,Y):－father(X,Z),parent(Z,Y).

Zは本体の中で存在束縛されているとみなせる。

計算としての反駁

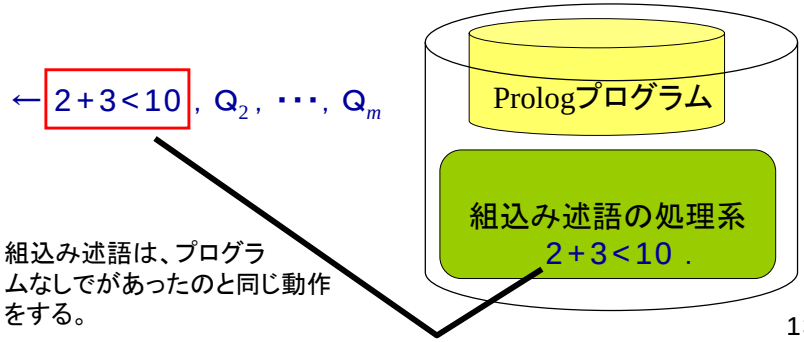
ゴールが変数Xを含む場合、

- 変数Xは、存在限定とみなせる。つまり、
  「…を成立たせるXはありますか？」というゴールとみなせる。



組込み述語

- DB内の節との導出のみでなく、組込みの述語(built-in predicate)によって計算が進められる。
- 組込みの述語: 頻繁に利用される述語、数値計算を行う述語、副作用を伴う述語(入出力、DB操作)



「P :ー Q_1, ・・・, Q_m.」の意味

- 「P :ー Q_1, ・・・, Q_m.」はいくつかの解釈が可能。
- 論理式として:
  - 条件Q_1, ・・・, Q_mが成立っているとき、Pが成立つ。
  - 逆に読めば、Pが成立つには、条件Q_1, ・・・, Q_mが成立っておればよい。
- 手続きとして:
  - 目標Pを達成するには、副目標Q_1, ・・・, Q_mを達成すればよい。

リスト

- Prologでもリストは重要なデータ構造である。
- リストは [1, 2, 3]という表現を持つが、LISPのconsと同様に、先頭要素と、先頭を除いた要素を結合する「.」(ドット)でも表現できる。
  [1, 2, 3]= .(1, [2, 3])=.(1, .(2, .(3, [])))
- Prologでは、[]をnilとは書かない。
- さらに、先頭と先頭以外を「|」で分けて書く記法がある。
  [1, 2, 3]= [1 | [2, 3]]= [1 | [2 | [3 | []]]]
- 次の書き方も可能。
  [1, 2, 3, 4, 5]= [1, 2, | [3, 4, 5]]= [1, 2, 3 | [4 ,5]]

項(term)とデータ構造

- Prologの基本データ構造は項(term)。
- 好きなように関数記号を組合わせて項を用いることができる。
  例 father(mother(taro))
  medcheck(taro, height(173),weight(62))
  どのように解釈するかは、ユーザーしだい。
- リストや数式も単なる項。
  [1,2,3] = .(1, .(2, .(3, [])))
  1+2*5 = +(1, *(2, 5))
- 見た目はよく似ているが、述語記号と関数記号は全く別物。
  wants(brother(taro), book).

組込み述語

- 数式
  - X is 数式 ： 数式を評価してXに単一化する。
    例 X is 2+3　　X is sqrt(2)
    数式はこのゴールに出会った時点で変数を含んでいてはならない。
  - 数式 ＜ 数式 ： 数式を評価して等号／不等号を成否を見る。＜,＞,＝＜,＞＝,＜＞,＝＝ など。
    例 3＊5 ＜ 2+3　　5－1＝:＝2＋3
- 副作用を伴う述語
  - 入出力 ： write(X), read(X)
- データベースの操作
  - assert：データベースへの節の追加
  - retract：データベースから節の削除

Prologの変数と大域的状態変化

- Prologには通常の変数の概念がない。
  例 P(X,Y):-Z is X＊2, W is Z+3, Z+W<Y.
- 値の代入をすることはできない。
- 変数は単一化するのみ。
- 一旦、単一化（具体化）されると他の値に単一化しなおすことはできない。

- 大域的な状態を記憶したい場合は、データベースにassert、retractする。
  例 assert(age(taro, 14)).
  　retract(age(taro, _)).
  　assert(age(taro, 15)).

プログラム例

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

- fact(0,1)
- fact(X,Y):-X1 is X-1, fact(X1,Y1),Y is Y1＊X.

実行順序：非論理的要素

prologではP∧Q＝Q∧P、P∨Q＝Q∨Pが成立たない。

- 節の順序　P∧Q≠Q∧P
  - fact(0,1)
  - fact(X,Y):-X1 is X-1, fact(X1,Y1),Y is Y1＊X.

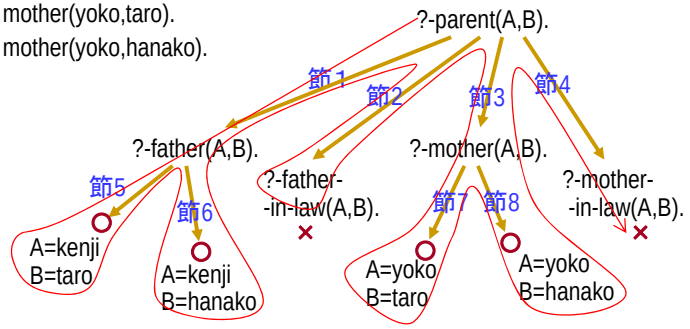
  の2つの節の順序は変えられない。
  SLD導出では、上から順に導出可能な節を探す。
- リテラルの順序：P∨Q≠Q∨P
  次の節の本体リテラルの順序は変えられない。
  - fact(X,Y):-X1 is X-1, fact(X1,Y1),Y is Y1＊X.

  SLD導出では左から導出を試みる。
  変数が具体化されていないと、isで失敗する。

バックトラック

節１　parent(X,Y):-father(X,Y).
節２　parent(X,Y):-father-in-law(X,Y).
節３　parent(X,Y):-moter(X,Y).
節４　parent(X,Y):-moter-in-law(X,Y).
節５　father(kenji,taro).
節６　father(kenji,hanako).
節７　mother(yoko,taro).
節８　mother(yoko,hanako).

節を上から下へ、リテラルを左から右へ探索。全解を探索することもできる。途中で失敗した場合に別の探索経路に戻ることをバックトラックという。



カットと実行順序の制御

- 次式は2つのケース場合わけで定義している。
  - abs(X,Y) :－ X>=0, Y is X.
  - abs(X,Y) :－ X<0,Y is -X.
- このときは、1つ目の節のX>=0の部分で成功すれば、2つ目の節へのバックトラックは不要。
- バックトラックさせないことを明示するため、！を用いる。
  - abs(X,Y) :－ X>=0, !, Y is X.
  - abs(X,Y) :－ X<0, !,Y is -X.

失敗による否定
Negation as failure

- ホーン節では、節の本体はすべて否定。
- したがって否定的意味を扱うことはできない。

- しかし、次のとおりnotを定義すると、おおよそ否定の意味になる。

not(P):－P, !, fail.
not(P).

Pが成功すれば、そこでバックトラックを打ち切り、失敗させる。
最初から失敗した場合は、バックトラックで2つ目の節を採用し、無条件で成功。

練習

- sum2n(N,S)で、S＝1＋2＋・・・＋Nを求める述語。
- exp(X,Y)で、Y＝2^Xを求める述語。
- length(L, X)で、リストLの長さをXに求める述語。
- sumlist (L, S)で、数のリストL中の数の合計をSに求める述語。

まとめ

論理型言語パラダイムの代表としてPrologを概観した。

- Prologでは
  - プログラム＝論理式の集合（理論）
  - プログラムの実行＝証明
- 非論理的要素も含む
  - 実行の順序（上から下、左から右）＝SLD導出
  - バックトラックとカット
- 項が基本のデータ構造