Love! Prolog

2009.4.4  勉強会

Naruhiko Ogasarara
Id: naruoga (@ Hatena-dialy, gmail, Twitter)
naru01 (wassr)

お詫び

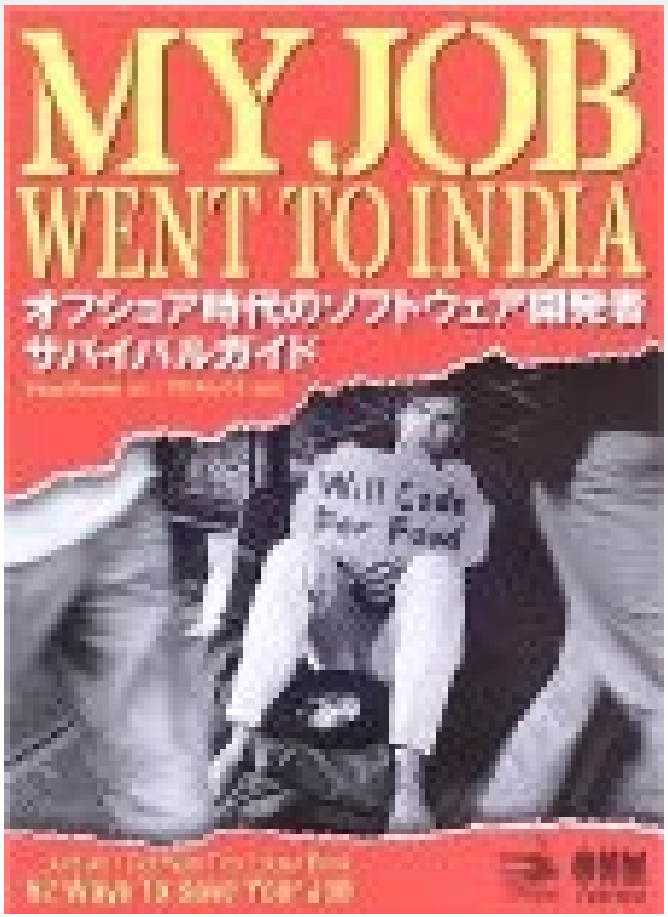
この資料の
前半部分は
別のプレゼンからの
思いっきり流用
です。

YouTube や
SlideShare や
オイラの日記で
見ちゃった人
すみません。

しかも
分量が
倍ぐらいに
増えてます (^^;)

さて、
気を
取り直して。

"My Job Went To India" 曰く



Chad Flower / でびあんぐる
オーム社
ISBN: 4274066592

「プログラマとして
生き残りたければ
いろんな
パラダイム
の言語を学べ」

C++ と
Java と
Perl と
Ruby と
Python と
PHP と……

それ全部
同じパラダイム
（手続き型言語）
ですから！

違う
パラダイム
？

関数型言語とか
ちょっと
流行ってるよね

Lisp/Scheme
とか
Haskell
とか
Ocaml
とか

でも
忘れないで

もう一つの
プログラム言語
パラダイム

論理型言語！

1980 年代
日本を席巻した
「第五世代
コンピュータ
構想」

みんな忘れたがってると思うが。

その中心に
あったのが
「人工知能 (AI)
言語
Prolog」

とゆことで
その魅力を
愛(AI)を以って
語ります。

Prolog
とは

おフランス
生まれの
小粋な言語

PROgramming In LOGic の略

ホントはフランス語なんだけどそんなん知らんわ。

openSUSE で使うには

有名な
処理系には
SWI-Prologと
Gnu-Prologが
あります。

私は
SWI-Prolog
を
使ってますが
理由は
特にないです (^^;)

インストールは
OBS で
ワンクリック
インストール！

実行ファイルは
/usr/bin/pl です。
拡張子も .pl
(Perl なんぞに
負けるな！)

さておき。

Prolog における プログラム とは

「論理」で
表現された
「知識」

「知識」に
問い合わせを
して
答えをもらうのが
Prolog の
プログラムの「実行」

例

```
男 ( なるひこ ).
男 ( まさと ).
女 ( あつこ ).
親 ( まさと , なるひこ ).
親 ( あつこ , なるひこ ).
人間 (X) :-  男 (X).
人間 (X) :-  女 (X).
父 (X,Y) :-  親 (X,Y),  男 (X).
母 (X,Y) :-  親 (X,Y),  女 (X).
```

誰かさんの家族構成がモロバレなのは気にしないでください。

なんじゃ
こりゃ？

順を追って
説明
しましょう。

基本概念 1
述語 (predicate)

男 (なるひこ)

↓

「なるひこ」は
「男」である

と読む

一般的には

$p(a_1,a_2,\ldots\ldots,a_n)$

は

「 $a_1,a_2,\ldots\ldots,a_n$ は

p（という関係）」

と読める

基本概念 2
事実 (fact)

述語にピリオドを
打てば
それが事実。

```
男 ( なるひこ ).
```

「なるひこは男、というのは（Prolog 内知識では）事実」という意味

基本概念 3
規則 (rule)

p_0 :- p_1,p_2,……,p_n.

は

「 p_1,p_2,……,p_n が

すべて真であれば

p_0 も真」という意味

p_0 :- p_1,p_2,……,p_n.

は

「 p_1,p_2,……,p_n が

すべて真であれば

p_0 も真」という意味

```
人間 (X) :-  男 (X).
.....
母 (X,Y) :-  親 (X,Y),  女 (X).
```

「Ｘが男なら、
Ｘは人間」
「ＸがＹの親で、Ｘが女なら、ＸはＹの母」

ほいじゃま
この
「プログラム」
を
実行して
みましょう

デモ

事実と
規則からなる
知識から
ゴールに適する
答えを
探索してくれる

では

「どんな風に
このプログラムが
動くか？」

を
お話し
します。

キーワードは
「ユニフィケーション
（単一化）」
と
「バックトラック」

先ほどの知識に
次のゴールを
投げ込んだとき

?- 母 (Mother, なるひこ).

男 (なるひこ).
男 (まさと).
女 (あつこ).
親 (まさと , なるひこ).
親 (あつこ , なるひこ).
人間 (X) :- 男 (X).
人間 (X) :- 女 (X).
父 (X,Y) :- 親 (X,Y), 男 (X).
母 (X,Y) :- 親 (X,Y), 女 (X).

?- 親 (Mother, なるひこ), 女 (Mother).

これが
ユニフィケーション
（単一化）

男 (なるひこ).
男 (まさと).
女 (あつこ).
親 (まさと , なるひこ).
親 (あつこ , なるひこ).
人間 (X) :- 男 (X).
人間 (X) :- 女 (X).
父 (X,Y) :- 親 (X,Y), 男 (X).
母 (X,Y) :- 親 (X,Y), 女 (X).

?- 親 (まさと , なるひこ), 女 (まさと).

これは別の
ユニフィケーション
（単一化）

男 (なるひこ).
男 (まさと).
女 (あつこ).
親 (まさと , なるひこ).
親 (あつこ , なるひこ).
人間 (X) :- 男 (X).
人間 (X) :- 女 (X).
父 (X,Y) :- 親 (X,Y), 男 (X).
母 (X,Y) :- 親 (X,Y), 女 (X).

ゴール前半に
対応する
候補を発見

?- 親 (まさと , なるひこ), 女 (まさと).

”女 (まさと)” に
対応する
知識がない
＝問い合わせ
失敗!!!

男 (なるひこ).
男 (まさと).
女 (あつこ).
親 (まさと , なるひこ).
親 (あつこ , なるひこ).
人間 (X) :- 男 (X).
人間 (X) :- 女 (X).
父 (X,Y) :- 親 (X,Y), 男 (X).
母 (X,Y) :- 親 (X,Y), 女 (X).

?- 親 (あつこ , なるひこ), 女 (あつこ).

男 (なるひこ).
男 (まさと).
女 (あつこ).
親 (まさと , なるひこ).
親 (あつこ , なるひこ).
人間 (X) :- 男 (X).
人間 (X) :- 女 (X).
父 (X,Y) :- 親 (X,Y), 男 (X).
母 (X,Y) :- 親 (X,Y), 女 (X).

(2)の問い合わせは失敗したので遡って新しい解を探す（バックトラック）

3'

?- 親 (あつこ , なるひこ), 女 (あつこ).

男 (なるひこ).
男 (まさと).
女 (あつこ).
親 (まさと , なるひこ).
親 (あつこ , なるひこ).
人間 (X) :- 男 (X).
人間 (X) :- 女 (X).
父 (X,Y) :- 親 (X,Y), 男 (X).
母 (X,Y) :- 親 (X,Y), 女 (X).

”女 (あつこ)” に
対応する
知識あり
＝問い合わせ
成功!!!

4’

?- 母 (Mother, なるひこ).
Mother = あつこ

男 (なるひこ).
男 (まさと).
女 (あつこ).
親 (まさと , なるひこ).
親 (あつこ , なるひこ).
人間 (X) :- 男 (X).
人間 (X) :- 女 (X).
父 (X,Y) :- 親 (X,Y), 男 (X).
母 (X,Y) :- 親 (X,Y), 女 (X).

バックトラックと
ユニフィケーション
が
Prolog の
真骨頂
なのです。

バックトラックを
うまく使うと
欲しい解を
全部得られる

「失敗駆動ループ」
ってのが
あるんですが
時間の都合で
割愛です（T_T)

話
変わりまして。

Lisp
ご存知な方
挙手
してください！

Lispといえば
List Processor
の略であり

リスト処理なら
任せろ!って
言語ですが

Prolog の
リスト処理も
ちょっと
すごいよ？

例題1:
append

二つのリストを
結合する

append([1,2,3],[4,5,6],X)

X = [1,2,3,4,5,6]

Prolog 版：

```
my_append([], A, A).
my_append([A1|A2], B, [A1|C]) :-
        my_append(A2, B, C).
```

Lisp 版：

```
(defun my_append (L1 L2)
  (if (null L1)
      L2
      (cons (car L1) (append (cdr L1) L2))))
```

例題２：
member

リストの中に要素が含まれるか調べる

member(1, [2,3,4])

fail.

Prolog 版：

```
my_member(A, [A|_]).
my_member(A, [_|C]) :-
    my_member(A, C).
```

Lisp 版：

```
(defun my_member (X L)
  (cond ((null L) nil)
        ((= (car L) X) t)
        (t (my_member X (cdr L)))))
```

共通点も
あります。

1. 再帰処理を つかっていること

2.「リストの先頭要素」と「それ以降」という表記があること

Prolog: [A|B]
Lisp: (cons (car L) (cdr L))

でも Prolog には Lisp にはない 特徴がある!

それは！

逆演算!

デモ

これは
Prologの表現が
「論理的に
どうある」を
記載するので

逆の答えが
得られるように
勝手に
なってしまう!
のです。

さて。

今まで敢えて

「知識」と
「事実」と
いう
言葉を使って
きましたが。

Prolog は
知識の
論理記述
だけにしか
使えない
言語なの？

否！

今までの
Prolog の解釈は
「宣言的」

それに対して
「手続的」
解釈も
可能

$p_0 :- p_1, p_2, \ldots\ldots, p_n.$

「$p_1, p_2, \ldots\ldots, p_n$ がすべて真であれば p_0 も真」:

宣言的解釈

p_0 :- p_1,p_2,……,p_n.

「p_0 という処理は
p_1,p_2,……,p_n を
実行すること」:
手続的解釈

宣言的に
考えた方が
Prolog らしい
ですが
手続的な
処理も書けます。

長いですね。
後もうちょいです。
許してください。

Prologの人工知能言語たる所以

「自己増殖機能」

自分自身で
事実や
規則
（＝プログラム）を
拡張して
いける

例えば
さっきの
知識空間に
こんなことを
してみましょう。

?- assert(親 (まさと , ちかげ)).
true.

?- assert((兄弟 (X,Y):- 親 (P,X), 親 (P,Y))).
true.

?- 兄弟 (X, なるひこ).
X = ' なるひこ ' ;
X = ' なるひこ ' ;
X = ' ちかげ ' ;
false.

```
?- listing.

:- dynamic' 親 '/2.

' 親 '(' まさと ', ' ちかげ ').

...

:- dynamic' 兄弟 '/2.

' 兄弟 '(A, C) :-' 親 '(B, A),' 親 '(B, C).

...

true.
```

assert という
組み込み述語を
使うことで
知識空間に
規則や知識を
追加できる！

つまり
プログラム
そのものを
ガンガン
拡張できる！

文字列で
コード作って
eval とか
クソだぜクソ!
（下品でごめん）

応用例：
Animals

質問に
答えることで
どんどん
動物の知識を
増やしていく
コンピュータソフト

例：

動物を一種類思い浮かべてください。
それを当ててみましょう。

それは人間ですね？ n
それはなんですか？ 象
象と人間を区別する質問は？　足は4本ですか？

ではもう一度。
動物を一種類思い浮かべてください。
それを当ててみましょう。

足は4本ですか？　y
それは象ですね？　n
それは何ですか？　馬
馬と象を区別する質問は？　鼻は短いですか？

…

動物と
質問の知識を
assert で
どんどん増やせば
いいだけだから
簡単！

というわりに
作って
こなかったのは
私の怠慢が
悪いんです＞＜

あとで
ブログで
公開するから
遊びにきてね。

そんなわけで
エレガントかつ
キャッチーな機能
たっぷりな
超高水準言語
Prolog。

あなたの
言語コレクションに
お加え
ください。

```
interest(prolog).
love(Someone,
                Something)
:-
  engineer(Someone),
  interest(Something).

?- love(you, prolog).
```

ご清聴
ありがとう
ございました。

参考 URL

- SWI-Prolog
  - GPL の Prolog 処理系
  - 様々なプラットフォームに対応
    - Debian 系 Linux なら apt-get install swi-prolog でインストールできるよ
  - Linux 版は X インタフェースやシェル呼出しも可
  - ドキュメントも充実
- お気楽 Prolog プログラミング入門
  - 日本語で書かれたチュートリアルとしてはかなりまとまっていると思います。オススメ。