Lisp, Prolog で N-Queen 問題を解く

工学部情報知能工学科 徳永 大輔

1.はじめに

Lisp、Prolog という人工知能の分野で良く使われるプログラミング言語を紹介し、簡単なパズルを解かせてみたいと思います。

2. N-Queen 問題

N-Queen 問題というパズルがあります。1回で上下左右斜めに何マスでも自由に動ける Queen という駒がチェスにはあります（将棋でいう飛車＋角の動きです）。N×Nのチェスの盤面にどの Queen 同士も1手で取り合えないようにN個の Queen をうまく配置する方法を考えるものです。いろいろな解決法があり、プログラミングの基本的な知識だけで作ることができるので、多くのプログラミング言語の入門書に掲載されています。例えば下の盤面、

```
1 2 3 4 5 6 7 8
1| | | | | |Q| | |
2| | |Q| | | | | |
3| | | | |Q| | | |
4| | | | | | |Q| |
5|Q| | | | | | | |
6| | | |Q| | | | |
7| |Q| | | | | | |
8| | | | | | | |Q|
```

は、8-Queen 問題の解の一つです。なぜなら、どの Queen も1手で他の Queen を取ることができないように 8 個の Queen が配置できているからです。

3. Lisp で解く

3.1 Lisp とは

Lisp の歴史は古く、1950 年代後半から 1960 年代始めにかけて MIT の J。 McCarthy らによって開発されました。当時は FORTRAN や ALGOL、COBOL 等の手続き型言語、数値処理型言語などと呼ばれる言語が作られ始めた頃で、Lisp はこれらの言語に対抗する言語として関数型言語、記号処理型言語などと呼ばれています。Lisp の利点としては、記号（symbol）を処理する

能力を持っている点、リスト(C 言語などでいう配列を強化したようなもので、可変長で入れ子(nest)構造をとれるデータの集合)を処理できる点(Lisp の語源は LISt Processor、LIStProgramming などと言われています)などがあります。これらの特長が人工知能の分野の特に記号処理分野、具体的にはエキスパートシステムや数式処理、定理証明、知識処理などにおいて、複雑なデータを処理するためのプログラミング言語として適していることから Lisp は人工知能のアルゴリズムやプログラムの開発の歴史とともに発展してきました。Lisp には MacLisp、InterLisp などかなりの数の方言が存在していましたが、これらの言語仕様の統一を目指して 1984 年に Guy Steele らによって作られた Common Lisp が現在の Lisp の標準的な仕様とされています。また、普段から emacs や mule を利用されている人は Lisp を知っていれば文書の編集に使うマクロのコマンドを自由に作成、定義することができるという利点もあります。

3.2 Lisp の起動

情報処理センターの自習室や演習室の O2 には GNU Common Lisp(以下 GCL)が、並列サーバ gaia には Allegro Common Lisp(以下 Allegro CL)が入っています。GCL は /usr/local/bin/ にパスを通して gcl、Allegro CL は /usr/acl4。3/bin/にパスを通して clim2xm¥_composer と入力すれば起動します。起動直後は GCL では、

>

Allegro CL では、

USER(1):

と表示されてユーザの入力待ちになります(コマンドラインです)。これで準備 OK です。終了するには、C-d(Control キーと d キーの同時押し)です。

3.3 Lisp による N-Queen 問題の計算

それでは、いよいよプログラミングに入りましょう。

- プログラムソース(文献[4] を一部修正) あらかじめ以下の queen1。l をファイルとして保存して、コマンドラインから、

(load "queen1。l")

と入力すればロードできます。

queen1.l

```
------------------------------------------------------------------------
-----
(defun queen1-board (n)
(do ((ans (queen1 n) (cdr ans)))
((eq ans nil))
(outp n (car ans))))

(defun queen1 (n)
(subqueen1 n nil n))

(defun outp (n list)
(do ((j 0 (+ j 1)))
((eq j n))
(princ #¥|)
(do ((k 1 (+ k 1)) (l (nth j list)))
((eq k (1+ n)))
(cond ((eq k l) (princ #¥Q) (princ #¥|))
(t (princ #¥Space) (princ #¥|))))
(princ #¥Linefeed))
(princ #¥Linefeed))

(defun subqueen1 (m b n) ;; 1 行目
(cond ((zerop m) nil) ;; 2 行目
((or (member m b) ;; 3 行目
(qp 1 b m)) ;; 4 行目
(subqueen1 (1- m) b n)) ;; 5 行目
(t (nconc (cond ((equal (length b) (1- n)) ;; 6 行目
(list (cons m b))) ;; 7 行目
(t (subqueen1 n (cons m b) n))) ;; 8 行目
(subqueen1 (1- m) b n))))) ;; 9 行目

(defun qp (k m n)
(cond ((null m) nil)
((equal (abs (- n (car m))) k) t)
(t (qp (1+ k) (cdr m) n))))
```

- 実行

実際に動かしてみましょう. コマンドラインで、

(queen1-board 8)

と、入力すると、

```
| | | | |Q| | | |
| | | | | | |Q| |
| |Q| | | | | | |
| | | | | |Q| | |
| | |Q| | | | | |
|Q| | | | | | | |
| | | |Q| | | | |
| | | | | | | |Q|
```

が続々と出てきましたね(8-Queen の場合 92 個)。これらはすべて 8-Queen 問題の解になっています。*queen1-board* の引数(今回は 8)を変えれば、5-Queen や 10-Queen などに対応できるようにしてあります。

- 解説

defun で定義された5つの関数を使います。処理の本体は *subqueen1* です。また、{¥tt qp} は今置こうとしている点から斜めの位置に他の Queen がないかをチェックするための関数です。また、

(subqueen1 8 nil 8)

を実行するとリスト形式で答が出てきます。それを盤面の形式にするために残り 3 つの関数を用意しました。

subqueen1 は現在の縦座標 m、現時点で確定している Queen の位置を表すリスト b、置きたい Queen の数 n(N-Queen の N と同値)、の 3 つの引数からなる関数です。subqueen1 は(n、n) の座標から数字を 1 ずつ減らしながら Queen を置くことが可能な位置を総当たりで発見してリストb に蓄えていく方法で実行されます。cond は C 言語の switch と同じ意味です。2 行

目はもし座標値 m が 0 なら関数が nil(空リスト)を返して終了することを意味します。3、4、5 行目は置こうとする Queen がすでに置き場所が確定している Queen からとれる位置(横、斜めの位置)にある場合、ここには置けないとわかるため、次の座標に処理を移します。具体的には、縦座標を 1 減らして再帰呼び出しします。6 行目の t は C 言語の default と同じ意味です。nconc は 6、7、8 行目の cond の返り値(リスト)と 9 行目の subqueen1 の返り値(リスト)を結合します。これによって答の候補がリストの形で蓄えられていきます。6 行目の cond 以下は、今から置く Queen が最後の 1 個ならリストを完成させて(cons も nconc 同様リストの結合ですが意味は大きく違います。ここでは詳しい説明はしません)そのリストを返り値にし、最後の 1 個じゃない時は現在までに確定している Queen の位置に入れて、リスト b の先頭に現在の縦座標値を入れて(結果的に次の横座標に処理を移すことになる)再帰呼び出しをします。縦座標が 0 になるか(縦を端から端までチェックが終った時)、最後の Queen が置かれる時(横を端から端までチェックが終った時)のいずれかを満たせば再帰呼出しが終了します。

4. Prolog で解く

4.1 Prolog とは

Prolog は 1970 年代初期にエジンバラ大学の R。 Kowalski らによって提案された考え方である PROgramming in LOGic を語源としており、その名の通り、論理数学の考え方をプログラミング言語として使おうという目的で作られたものです。Prolog は論理型言語と呼ばれ、手続き型言語では記述が困難な事物間の関係や性質を論理式の形で容易に記述、利用することが最大の長所となっています。従って、Prolog のプログラムは論理式の集まりになっています。Lisp 同様、Prolog は記号処理などの非数値演算やリスト処理を行なうことができ、問題軽決、発見的探索、エキスパートシステムなど人工知能の分野で応用がなされています。1974 年にマルセイユ大学の Colmerauer らによって開発され、 現在ではエジンバラ大学の DEC-10 構文則が標準的な仕様とされています。

4.2 Prolog の起動

並列サーバ gaia の /usr/local/bin/ にパスを通し、prolog と打つと QuintusProlog が起動します。起動すると、

```
| ?-
```

というコマンドラインが表示されます。終了は Lisp と同様、C-d です。

4.3 Prolog による N-Queen 問題の計算

- プログラムソース（文献 [1] を一部修正）、以下のソースを queen2.pl という名前で保存して、Prolog を起動し、コマンドラインで、

[queen2].

と打ち込んで（ピリオドも入れる）ロードします。

queen2.pl

```
------------------------------------------------------------------------------
-----
queen2(N, S) :- % 1 行目
gen(1, N, Dxy), % 2 行目
Nu1 is 1 - N, Nu2 is N - 1, % 3 行目
gen(Nu1, Nu2, Du), % 4 行目
Nv2 is N + N, % 5 行目
gen(2, Nv2, Dv), % 6 行目
sol(S, Dxy, Dxy, Du, Dv), % 7 行目
writesol(S,N). % 8 行目

gen(N, N, [N]).
gen(N1, N2, [N1|List]) :-
N1<N2,
M is N1+1,
gen(M,N2,List).

sol([], [], _, _, _). % 1 行目
sol([Y|Ylist], [X|Dx1], Dy, Du, Dv) :- % 2 行目
del(Y, Dy, Dy1), % 3 行目
U is X-Y, % 4 行目
del(U, Du, Du1), % 5 行目
V is X+Y, % 6 行目
del(V, Dv, Dv1), % 7 行目
sol(Ylist, Dx1, Dy1, Du1, Dv1). % 8 行目

del(A, [A|List], List).
del(A, [B|List], [B|List1]) :-
del(A, List, List1).
```

```
writesol([],_).
writesol([Sol|Solrest],N) :- !,
write('|'),
writes(Sol,N),nl,
writesol(Solrest,N).

writes(_,0).
writes(Sol,N) :-
Sol =:= 1,
write('Q|'),
Solnew is Sol - 1,
Nnew is N - 1,
writes(Solnew,Nnew).
writes(Sol,N) :-
Sol =¥= 1,
N > 0,
write(' |'),
Solnew is Sol - 1,
Nnew is N - 1,
writes(Solnew,Nnew).
```

- 実行

コマンドラインで

```
queen2(8, Ans).
```

と打つと、

```
|Q| | | | | | | |
| | | | |Q| | | |
| | | | | | | |Q|
| | | | | |Q| | |
| | |Q| | | | | |
| | | | | | |Q| |
```

```
| |Q| | | | | | |
| | | |Q| | | | |
```

Ans = [1,5,8,6,3,7,2,4]

と、出てきます。これが 8-Queen 問題の解の一つになっています。ここで、

;

を入力すると別解が次々に出てきます（これについては解説で詳しく述べます）。Lisp 同様、queen2 の引数の 8 を 5 や 10 に変えれば、5-Queen や 10-Queen 問題も解けるようになっています。

- 解説

このアルゴリズムは 4 つの座標系（横 x、縦 y、右上がりの対角線 u、左上がりの対角線 v)を作ることから始まります。当然、対角線の座標の値は横、縦の値から求めることができます。

u = x − y
v = x + y

とすると、4つの座標系の定義域 D は、n 個の Queen を置く問題の時には、

D_x = 1,2,・・・,n
D_y = 1,2,・・・,n
D_u = 1−n,2−n,・・・,n−1
D_v = 2,3,・・・,2n

となります。このアルゴリズムでは横、縦の値を決定するたびにこれらのリストから該当する座標値 4 つ（横、縦、右上がりの対角線、左上がりの対角線）を順に取り除いていくものです。もし、取り除こうとした座標値がリスト内にない場合は、今置こうとしている Queen がすでに置かれた Queen から縦、横、斜めのどこかに位置していることを指し、その点には Queen が置けないことになります。queen2 の 2 行目で横、縦の定義域をリストとして作成し、3、4 行目で右上がり、5、6 行目で左上がりの対角線の定義域をリストとして作成しています。gen(X、Y、Z) は X から Y まで 1 ずつ増える数字のリストを Z に入れるものです（Prolog では大文字で始まる語句は変数、小文字で始まる語句は定数です）。sol は処理アルゴリズムの本体です。1 行目は横、縦のリストが空になる（n 個の Queen をすべて置き終えた）時に再帰呼び出しを終了することを意味します。

2 行目の [Y|Ylist] は Y がリストの先頭、 Ylist がリストの残りを指します。つまり、リストの先頭の値 Y だけを取り出して本文中で使い、残り Ylist は次に再帰呼び出し(8 行目)しています。X についても同様です。3 行目は横座標リスト Dy から Y を外すものです(del(X、Y、Z) は X をリスト Y から取り除いたらリスト Z ができることを意味します)。同様に、4、5 行目で座標系 Du、6、7 行目で座標系 Dv に同じ処理を行ない、残ったリストで再帰呼出ししています。 writesol と writes は結果をリストから盤面に変えて出力するためのものです。

ここで、Prolog が他のプログラム言語と決定的に違う点は、それぞれの関係は定義しましたが、入力がどれで出力がどれというのを決める必要がない点です。Prolog は入力、出力を常に考えなければいけない手続き型言語とは大きく違います。この例をプログラム内の del を使って説明します。queen2。pl をロードした段階で、

del(a,[a,b,c],Z)。

と入力すると、

Z = [b,c]

と出ます。これの意味は``リスト [a、b、c] から a を除いた結果 Z として考えられる組合せを全て出せ"です。つまり Z として考えられる値(リスト)をパターンマッチングによって全て出してくれます(この場合は[b、c] しかあり得ないので ; を入れても no が返ってくる、即ち、他の候補がないことを意味します)。また、

del(X、[a、b、c]、[b、c])。

と入力すると、

X = a

と出ます。これの意味は``リスト [a、b、c] から何を除いたら [b、c]が得られるか、考えられる組合せを全て出せ"です。また、

del(X、[a、b、c]、Z)。

と入力すると、

X = a、
Z = [b、c] ;

X = b、
Z = [a、c] ;

X = c、
Z = [a、b] ;

と出ます(; は随時入力)。これの意味は``リスト [a、b、c] の要素を分ける時に考えられる組合せを全て出せ"です。つまり、Prolog は関係を与えておけば、既知の事実(この場合リスト [a、b、c]j から、関係を満たす組合せを全て捜し出してくれます(これを Prolog ではパターンマッチングと呼びます)。Prolog はこのように一定の条件(N-Queen ならば縦、横、斜めの位置関係にない、といった条件)を満たす非数値の組合せ(リスト)を探すのに適したプログラム言語といえます。馴れないうちは簡単な命令を作るのにも戸惑うでしょう。また、ここでは説明しませんが Prolog には !(カット)という重要な概念があります。ここでもかなり戸惑うことでしょう。しかし、これらを乗り越えられた時には新しいプログラミング言語の概念を発見できることでしょう。

5. おわりに

C 言語や BASIC、PASCAL などを普段から使ってる方にとっては、Lisp は括弧だらけでわかりにくい、Prolog は直感的に理解はできるが何か作れと言われると困る、などと思われたのではないでしょうか。いずれも C 言語や BASIC などの手続き型言語(Procedural Language)とは考え方や記述が根本的に違うので、とっつきにくい面は多少なりとあるとは思います。今回は N-Queen 問題というちょっと本格的な問題解決を紹介したので、更にわかりにくかったかも知れませんが、本文を読んで興味を持たれた方は図書館などで入門書などを借りて1から始めるとよいでしょう。Lisp、Prolog ともにリスト処理など一度使いだすとやめられないような``奥の深さ"を随所に持っているプログラミング言語です。C 言語で segmentation fault を連発してしまう人、ポインタが恐くて使えない人は一度試してみてはどうでしょうか。

参考文献

[1] Bratko、 I. 著、 安部憲広 訳、 Prolog への入門、 近代科学社、 1990.

[2]Winston、 P. H.、 Horn、 B. K. P. 著、 白井良明、 安部憲広 訳、 LISP、 培風館、 1982.

[3]塚本龍男 著、 わかる :- Prolog、 共立出版、 1989.

[4]中西正和 著、 Lisp 入門、 近代科学社、 1985.

[5]湯浅太一、 萩谷昌己 著、 Common Lisp 入門、 岩波コンピュータサイエンス、 1986.