第13講 人工知能言語

- 人工知能プログラミングの特徴
- 人工知能言語の条件
- 人工知能言語の種類
- Lisp言語
- Prolog言語

人工知能プログラミングの特徴

- 対象世界の非構造･非定義性(ill-structured/ill-defined)
  - 知能･知識などのその構造が複雑、あいまい、未解明なものが対象となっている
- 演算や操作の複雑性･多様性
  - 複雑なパターンマッチング、演繹･帰納･類推などの多様な推論処理が行える
- 制御の非決定性･柔軟性
  - 再帰処理とバックトラック、さまざまな発見的な制御機能が備えられている

人工知能言語

- 問題、オペレータ、知識の記述が容易である
  - 記号による問題や知識を表現する言語には、LispとPrologがある
- 演繹機能の定義と命題表現が容易である
  - 人工知能の推論は命題を記号で表現し、三段論法による演繹推論をおおこなう
- パターンマッチング機能がある
  - 記憶しているデータを検索し、プロセスを制御していく機能は知識ベース構築に不可欠
- 再帰処理やバックトラックなどの融通性のある制御構造をもつ
  - 再帰、バックトラックともに、効率的な探索には不可欠な機能
- 関連する情報をグループ化する機能をもつ
  - フレームのような複雑な知識構造を記述でき、ひとつにまとまった事象として検索できる機能が必要

人工知能言語の種類

- IPL-V (Newell, Simon, Shaw) 1957, 定理証明
- LISP (McCarthy) 1958, 記号処理
- PLANNER (Hewitt) 1968, 計画立案、定理証明
- CONNIVER (Sussman, McDermott) 1972, 上記の改良版
- Prolog (Colmerauer) 1974, 知識表現
- FOL (Weyrauch) 1979, 一階述語論理

プログラミング言語の分類

- 手続き型言語 (Fortran)
  - 問題を処理する手順の系列として表現
- 関数型言語 (Lisp)
  - 問題を関係する関数間のデータの入出力関係によって表現
- 論理型言語 (Prolog)
  - 問題を処理する要素間の論理的関係で表現

Lisp

- Lisp (List processor)
  - 1957年 MITのJ. McCarthyにより開発
- 関数型言語の一種
  - 論理の一種であるラムダ計算 (Lambda calculus)を表現する言語
  - 「リスト」を基本データ構造とし、これを操作する記号処理プログラミング言語
  - 関数定義によって記述され、その定義自身がリスト形式で記述
  - 操作されるデータと関数がともにリスト形式で統一的に扱える
  - 米国の大学や研究機関で人工知能という分野ではぐくまれてきた

Lispプログラム

- 階乗計算

```
(defun fact (n)
  (cond ((= n 1) 1)
        (T (* n (fact (sub n 1))))))
```

Lispプログラム

- ハノイの塔

```
(defun hanoi (n x y z)
  (cond ((= n 1)  (move 1 x y))
        (T (progn (hanoi (sub n 1) x z y)
            (move n x y)
            (hanoi (sub n 1) z y x)
            nil))))
(defun move (k from to)
   (print (list ‘move k ‘from from ‘to to)))
```

Prolog

- Prolog (Programming in Logic)
  - 1974年 マルセイユ大学のA. Colmerauerにより開発
- 論理型言語の一種
  - ある論理式が真であることを融合法で証明する場合には、論理式は節（clause）の形式に変換されるので、知識をはじめから節の形式で表現しておけば、証明は節を使っておこなうことができる
  - 知識を（ホーン）節形式で表現し、融合法で計算をおこなうプログラミング言語
  - 計算することは論理式を証明することであるとみなす

Prologプログラム

- 階乗計算

```
fact (0,1).
fact (N, M) :- N1 is N –1, fact (N1, M1),
                M is N * M1.
```

Prologプログラム

- ハノイの塔

```
hanoi(1, A, B, _) :-
        move(1, A, B).
hanoi(N, A, B, C) :-
        N1 is N – 1,
        hanoi(N1, A, C, B),
        move(N, A, B),
        hanoi(N1, C, B, A).
move(N, X, Y) :-
        write([move, N, from, X, to, Y]),
        nl.
```