データベースの操作

副作用をもつ述語

Prolog のプログラムは宣言的に記述された一種のデータベースとみなすことができる．節の追加/削除はデータベースの更新に相当する．

副作用：実行中にデータベースが変わるので，デバッグ時には要注意．listing(Predicate) によってデータベースの更新をチェックする．(Predicate には述語名を代入する．)

デバッグ時はいったん halt して Prolog を再起動する方が無難．

組込み述語 assert と retract

- 節の追加 assert(C)
  C と単一化可能な節をデータベースの最後に追加する
- 節の追加 asserta(C)
  C と単一化可能な節をデータベースの最初に追加する
- 節の削除 retract(C)
  C と単一化可能な節のうち，データベースの先頭のものを削除する
- 節の削除 retractall(H)
  H と単一化可能なヘッドをもつ節すべてをデータベースから削除する

引数の数が Arity で述語名が Predicate である述語の定義節の冒頭に以下の 1 行を加えておく．

```
:- dynamic Predicate/Arity.
```

これによって，この述語はプログラム実行中に追加/削除が可能になる．

また，実行環境 (インタプリタ) において，

```
?- listing(fruit).
```

と入力すると，fruit の定義節のみを表示することができるので，現在のデータベースの内容確認に使用するとよい．

使い方の例

```
% プログラム                % 実行環境
:- dynamic fruit/0.         ?- assert(fruit:-orange).
fruit :- apple.             ?- retract(fruit:-G).
fruit :- lemon.                % ゴール自体が変数と単一化されることに注意
```

練習問題

1. 以下のようなデータベースから above の定義の第 1 節を below(pencil,bicycle) に変更するような述語 a2b のプログラムを作成せよ．正常に動いていることを listing によって確認せよ．まず変数を使わずに記述し，次に変数を使った記述方法を考えよ．

```
above(bicycle,pencil).
above(bird,sandglass).
above(camera,butterfly).
above(apple,fish).
```

2. 上のようなデータベースから above(X,Y) となっている節をすべて below(Y,X) という節に変更するような述語 above_to_below のプログラムを作成せよ．再帰的に定義すること．正常に動いていることを listing によって確認せよ．

3. 以下のようにデータベースがある．与えられたリスト L (ただし空リストではないとする) に書かれたすべての果物がこのデータベースにあるかどうかを判定する述語 contained(L) のプログラムを作成せよ．たとえば，contained([apple,banana]) は true となり，contained([orange]) は false となる，(この問題の解答では assert/retract は不要.)

```
fruit(banana).
fruit(apple).
fruit(grape).
```

4. 以下のようにデータベースがある．与えられたリスト L (ただし空リストではないとする) に書かれたすべての品物がこのデータベースにあるかどうかを判定する述語 contained2(L) のプログラムを作成せよ．たとえば，contained2([fruit(apple),veg(carrot)]) は true となり，contained2([fruit(orange)]) は false となる，(この問題の解答では assert/retract は不要.)

```
fruit(banana).
fruit(apple).
veg(carrot).
veg(spinach).
can(spam).
```

演習問題 (r8)

データベース部分も含むプログラム全体を添付すること.

(1) X が Y の右にある関係 right(X,Y) を使って書かれたデータベースを Y が X の左にある関係 left(Y,X) を使ったものに変更するような述語 update_db のプログラムを作成せよ．ただし，right の定義節がいくつあっても (有限個なら) 対応できるように再帰的に定義すること．たとえば,

```
right(apple,orange).
right(lemon,apple).
```

というデータベースは

```
left(orange,apple).
left(apple,lemon).
```

のように変更される.

(2) parent(X,Y), male(X), female(X) という 3 つの関係が定義されているデータベースにおいて，pam は X,Y いずれにも出現する可能性があるものとする．このとき，pam に関するデータをすべて削除する述語 del_all のプログラムを作成せよ．r1_2 で作成したデータベースで正常に動くことを確かめよ.

(3) 商店においてある品物のデータベースが以下のように定義されているとする．goods(X,Y) は，商品 X が Y の状態であることを示し，Y のとる値は新しい (new) か古い (old) のいずれかである.

```
goods(riceball,new).
goods(lunchbox,new).
goods(sandwitch,new).
goods(milk,new).
goods(yogult,old).
goods(tea,old).
```

購入品のリスト L (必ず 1 つ以上の品を含む) に書かれたすべての商品について新しい物が存在するかどうかを判定する述語 check(L) のプログラムを作成せよ．たとえば，check([sandwitch,milk]) は true となり，check([riceball,tea]), check([cheese]) は false となる．なお，この問題では assert/retract は使用しない.

(4) STRIPS 型プラニングでは，状態は環境モデルで表され，操作を適用すると環境モデルが変更される．プラニングでは，初期状態から目標状態に変更できるような操作の列を見つける．ここでは，環境モデルをデータベースとして記述し，操作をプログラムとして実装する．そして，環境モデルに対して操作を適用するとデータベースが変更されることを確認する．
環境モデルを変更させる操作は一般に,（操作が適用可能かどうかを判定する）条件リスト CL,（操作が適用されると成立しなくなる状態の要素を表す）削除リスト DL,（操作が適用されると新たに成立する状態の要素を表す）追加リスト AL を用いて以下のように定義される.

apply_operation(CL,DL,AL) :- cond_list(CL), del_list(DL), add_list(AL).

まず，cond_list(CL), del_list(DL), add_list(AL) をそれぞれ定義せよ.

「床 (floor) にいたオブジェクト X が隣のオブジェクト Y の上にのぼる」ことを表す操作 climb(X,Y) に対して，CL, DL, AL をそれぞれ [on(X,floor),next_to(X,Y),loadable(Y)], [on(X,floor),next_to(X,Y)], [on(X,Y)] と表現する．

次に，具体化した CL, DL, AL に apply_operation を適用することで climb(X,Y) を定義せよ．

さらに，環境モデル 1 がデータベースとして与えられるとき，環境モデル 1 の変化を確認することで正当な動きをすることを確かめよ．

(5) 環境モデル 2(参考資料の例) において，以下の操作 gotod(DX), passd(DX,RY), reach を STRIPS 型で定義し，これらを順に適用したときの環境モデル 2 の変化を確認せよ．ただし，box とドアとは十分離れているとする．

- gotod(DX)
  部屋の任意の位置にいる robot が別の部屋とつながったドア DX のそばにいくことを表すオペレータ
- passd(DX,RY)
  あいているドア DX のそばにいた robot が DX を通過して部屋から部屋 RY に移動することを表すオペレータ
- reach
  ドアのそばにいた robot が 同一部屋内の box のそばに移動することを表すオペレータ

(6) 解答例 r7_5 についてレポートせよ．

環境モデル 1

$on(c3po, floor)$
$on(r2d2, floor)$
$loadable(steelbox)$
$next\_to(c3po, steelbox)$
$next\_to(r2d2, door)$

環境モデル 2

$inroom(robot, room1)$
$inroom(box, room2)$
$status(door, open)$
$connect(door, room1, room2)$
$connect(door, room2, room1)$

意味

$loadable(X)$: X の上に何かをのせることができる
$on(X, Y)$: X は Y の上にいる
$next\_to(X, Y)$: X は Y の隣にいる (上にはいない)

$inroom(X, RX)$: X が部屋 RX に存在する
$connect(D, RX, RY)$: ドア D により
　　部屋 RX と部屋 RY がつながっている
$status(D, S)$: ドア D は S の状態である
$next\_to(X, Y)$: X は Y の隣にいる (上にはいない)