ENSEIRB Première année 2002/2003

Programmation Fonctionnelle : feuille 7

"General Problem Solver" d'après P. Norvig
AI Programming, M. Kaufmann publishers, 1992, Chapitre 4

Exercice 7.1

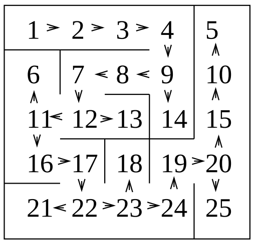
Considérer le source du "*General Problem Solver*" présenté en cours et disponible sur la page des TDs.

1. L'essayer et le tracer avec les requêtes du cours sur le domaine `*school-ops*`.
2. Implémenter une fonction permettant d'ajouter des règles.
3. L'utiliser pour faire varier le comportement du système.

Exercice 7.2

Utilisation sur un autre domaine.

1. Soit le labyrinthe suivant :



   Construire l'ensemble des règles qui le décrivent (indice: construire des fonctions qui génèrent les règles à partir d'un minimum d'information).

2. Faire des requêtes de chemins.

Exercice 7.3

Quelques défaillances de l'implémentation utilisées ci-dessus:

1. Trouver des cas qui font boucler GPS.
2. Quel est le problème lié à l'appel suivant :

```
(gps '(son-at-home car-needs-battery have-money have-phone-book)
     '(have-money son-at-school)
     *school-ops*)
```

3. Quel est le problème lié à l'appel suivant :

```
(gps '(son-at-home car-needs-battery have-money have-phone-book)
     '(son-at-school have-money)
     *school-ops*)
```

Exercice 7.4

Vers une meilleure implémentation :

1. Modifier le programme de telle sorte qu'à la fin d'un calcul tous les buts initiaux soient valides, c'est-à-dire qu'il n'y ait pas de séquentialité sous-jacente à la résolution d'un ensemble de buts.
2. Modifier le programme de telle sorte qu'une condition puisse être une liste (i.e. sous forme de prédicats "à la Prolog").
3. Modifier le programme de telle sorte que l'ensemble des opérations valides puissent être affecté extérieurement à `gps`.
4. Modifier le programme de telle sorte que sa sortie ne soit plus un simple "`solved`" en cas de réussite, mais une liste des actions effectuées (indice: toute action a pourra posséder dans son `add-list` la condition `(EXECUTING a)`).
5. Modifier le programme de telle sorte que les boucles deviennent impossibles (indice: il faut garder en mémoire la liste des buts en train d'être résolus).
6. Modifier le programme de telle sorte que la recherche ne se fasse plus sur chacun des buts séparément (indice: il faut se débarrasser de la variable globale `*state*` et gérer des états courants locaux).

```lisp
;;; -*- Mode: Lisp; Syntax: Common-Lisp; -*-
;;; Code from Paradigms of Artificial Intelligence Programming
;;; Copyright (c) 1991 Peter Norvig

;;;from auxfns.lisp ==============================
(defun find-all (item sequence &rest keyword-args
                 &key (test #'eql) test-not &allow-other-keys)
  "Find all those elements of sequence that match item,
  according to the keywords.  Doesn't alter sequence."
  (if test-not
      (apply #'remove item sequence
             :test-not (complement test-not) keyword-args)
      (apply #'remove item sequence
             :test (complement test) keyword-args)))

;;; File gps1.lisp: First version of GPS (General Problem Solver)
(defvar *state* nil "The current state: a list of conditions.")

(defvar *ops* nil "A list of available operators.")

(defstruct op "An operation"
  (action nil) (preconds nil) (add-list nil) (del-list nil))

(defun GPS (*state* goals *ops*)
  "General Problem Solver: achieve all goals using *ops*."
  (if (every #'achieve goals) 'solved))

(defun achieve (goal)
  "A goal is achieved if it already holds,
  or if there is an appropriate op for it that is applicable."
  (or (member goal *state*)
      (some #'apply-op
            (find-all goal *ops* :test #'appropriate-p))))

(defun appropriate-p (goal op)
  "An op is appropriate to a goal if it is in its add list."
  (member goal (op-add-list op)))

(defun apply-op (op)
  "Print a message and update *state* if op is applicable."
  (when (every #'achieve (op-preconds op))
    (print (list 'executing (op-action op)))
    (setf *state* (set-difference *state* (op-del-list op)))
    (setf *state* (union *state* (op-add-list op)))
    t))

;;;The historical domain example  ==============================
```

```
(defparameter *school-ops*
  (list
    (make-op :action 'drive-son-to-school
         :preconds '(son-at-home car-works)
         :add-list '(son-at-school)
         :del-list '(son-at-home))
    (make-op :action 'shop-installs-battery
         :preconds '(car-needs-battery shop-knows-problem shop-has-money)
         :add-list '(car-works))
    (make-op :action 'tell-shop-problem
         :preconds '(in-communication-with-shop)
         :add-list '(shop-knows-problem))
    (make-op :action 'telephone-shop
         :preconds '(know-phone-number)
         :add-list '(in-communication-with-shop))
    (make-op :action 'look-up-number
         :preconds '(have-phone-book)
         :add-list '(know-phone-number))
    (make-op :action 'give-shop-money
         :preconds '(have-money)
         :add-list '(shop-has-money)
         :del-list '(have-money))))
```