МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Тверской государственный технический университет»

(ТвГТУ)

Кафедра «Программного обеспечения»

**Отчёт по лабораторной работе №5**

по дисциплине “Системы искусственного интеллекта”

Выполнил: студент группы

ПИН-17.06

Завгороднев Е.Ю

Проверил:

Мальков А. А.

Тверь 2021

# Задача

Задача о миссионерах и каннибалах является классической задачей ИИ: трем миссионерам и трем каннибалам необходимо переправится на противоположный берег реки с помощью двуместной лодки. При этом число каннибалов на любом берегу не должно превышать числа миссионеров (иначе первые съедят вторых).

1. Построить для данной задачи полное дерево поиска.

2. Разработать программу решения задачи, используя среду CLIPS. Программа должна содержать три модуля: основной (MAIN), контроля ограничений (CONSTRAINTS) и вывода решения (SOLUTION).

2.1. Модуль MAIN должен содержать следующие конструкции:

· объявление модуля MAIN, включая объявления экспорта шаблона состояния (status) и глобальных переменных числа миссионеров (initial-miss) и каннибалов (initial-cann);

· определение шаблона факта-состояния (status);

· определение глобальных переменных initial-miss и initial-cann (задание их значений);

· определение факта исходного состояния initial-positions;

· определение факта вместимости лодки;

· определение функции вывода сообщения move-string;

· определение правил генерации пути в пространстве состояний.

2.2. Модуль CONSTRAINTS должен содержать:

· объявление модуля CONSTRAINTS, включая импорт из модуля MAIN шаблона status;

· определение правила, срабатывающего на запрещенные состояния, когда каннибалы могут съесть миссионеров;

· определение правила, срабатывающего в ситуации зацикливания процесса поиска (circular-path).

2.3. Модуль вывода решения SOLUTION должен содержать:

· объявление модуля SOLUTION, включая импорт из модуля MAIN шаблона status и глобальных переменных initial-miss и initial-cann;

· определение шаблона решения – последовательности перемещений (moves);

· определение правила распознавания целевого состояния (goal-test);

· определение правила построения решения (build-solution);

· определение правила вывода решения на экран (print-solution).

3. Выполните программу в пошаговом режиме, проанализируйте и объясните ход процесса поиска решения. В отчете необходимо привести трассу поиска решения.

# Листинг

(defmodule MAIN

(export deftemplate status)

(export defglobal initial-missionaries)

(export defglobal initial-cannibals)

)

(deftemplate MAIN::status

(slot shore-1-miss (type INTEGER) (range 0 ?VARIABLE))

(slot shore-1-cann (type INTEGER) (range 0 ?VARIABLE))

(slot shore-2-miss (type INTEGER) (range 0 ?VARIABLE))

(slot shore-2-cann (type INTEGER) (range 0 ?VARIABLE))

(slot boat-location (type SYMBOL) (allowed-symbols shore-1 shore-2))

(slot search-depth (type INTEGER) (range 1 ?VARIABLE))

(slot parent (type FACT-ADDRESS SYMBOL) (allowed-symbols no-parent))

(slot last-move (type STRING))

)

(defglobal MAIN ?\*initial-missionaries\* = 3

?\*initial-cannibals\* = 3

)

(deffacts MAIN::boat-information

(boat-can-hold 2)

)

(deffacts MAIN::initial-positions

(status (search-depth 1)

(parent no-parent)

(shore-1-miss ?\*initial-missionaries\*)

(shore-1-cann ?\*initial-cannibals\*)

(shore-2-miss 0)

(shore-2-cann 0)

(boat-location shore-1)

(last-move "No move"))

)

(deffunction MAIN::move-string(?miss ?cann ?shore)

(switch ?miss

(case 0 then

(if (eq ?cann 1)

then (format nil "Move 1 cannibal to %s.%n" ?shore)

else (format nil "Move %d cannibals to %s.%n" ?cann ?shore)))

(case 1 then

(switch ?cann

(case 0 then

(format nil "Move 1 missionary to %s.%n" ?shore))

(case 1 then

(format nil "Move 1 missionary and 1 cannibal to %s.%n" ?shore))

(default then

(format nil "Move 1 missionary and %d cannibals to %s.%n"

?cann ?shore))))

(default

(switch ?cann

(case 0 then

(format nil "Move %d missionaries to %s.%n" ?miss ?shore))

(case 1 then

(format nil "Move %d missionaries and 1 cannibal to %s.%n"

?miss ?shore))

(default then

(format nil "Move %d missionary and %d cannibals to %s.%n"

?miss ?cann ?shore)))))

)

(defrule MAIN::shore-1-move

(declare (salience 20))

?node <- (status (search-depth ?num)

(boat-location shore-1)

(shore-1-miss ?s1m)

(shore-1-cann ?s1c)

(shore-2-miss ?s2m)

(shore-2-cann ?s2c)

)

(boat-can-hold ?limit)

=>

(duplicate ?node

(search-depth =(+ 1 ?num))

(parent ?node)

;(shore-1-miss =(- ?s1m 0))

;(shore-2-miss =(+ ?s2m 0))

(shore-1-cann =(- ?s1c 1))

(shore-2-cann =(+ ?s2c 1))

(boat-location shore-2)

(last-move =(move-string 0 1 shore-2)))

(duplicate ?node

(search-depth =(+ 1 ?num))

(parent ?node)

;(shore-1-miss =(- ?s1m 0))

;(shore-2-miss =(+ ?s2c 0))

(shore-1-cann =(- ?s1c 2))

(shore-2-cann =(+ ?s2c 2))

(boat-location shore-2)

(last-move =(move-string 0 2 shore-2)))

(duplicate ?node

(search-depth =(+ 1 ?num))

(parent ?node)

(shore-1-miss =(- ?s1m 1))

(shore-2-miss =(+ ?s2m 1))

;(shore-1-cann =(- ?s1c 0))

;(shore-2-cann =(+ ?s2c 0))

(boat-location shore-2)

(last-move =(move-string 1 0 shore-2)))

(duplicate ?node

(search-depth =(+ 1 ?num))

(parent ?node)

(shore-1-miss =(- ?s1m 2))

(shore-2-miss =(+ ?s2m 2))

;(shore-1-cann =(- ?s1c 0))

;(shore-2-cann =(+ ?s2c 0))

(boat-location shore-2)

(last-move =(move-string 2 0 shore-2)))

(duplicate ?node

(search-depth =(+ 1 ?num))

(parent ?node)

(shore-1-miss =(- ?s1m 1))

(shore-2-miss =(+ ?s2m 1))

(shore-1-cann =(- ?s1c 1))

(shore-2-cann =(+ ?s2c 1))

(boat-location shore-2)

(last-move =(move-string 1 1 shore-2)))

)

(defrule MAIN::shore-2-move

(declare (salience 20))

?node <- (status (search-depth ?num)

(boat-location shore-2)

(shore-1-miss ?s1m)

(shore-1-cann ?s1c)

(shore-2-miss ?s2m)

(shore-2-cann ?s2c))

(boat-can-hold ?limit)

=>

(duplicate ?node

(search-depth =(+ 1 ?num))

(parent ?node)

(shore-1-miss =(+ ?s1m 0))

(shore-2-miss =(- ?s2m 0))

(shore-1-cann =(+ ?s1c 1))

(shore-2-cann =(- ?s2c 1))

(boat-location shore-1)

(last-move =(move-string 0 1 shore-1)))

(duplicate ?node

(search-depth =(+ 1 ?num))

(parent ?node)

(shore-1-miss =(+ ?s1m 0))

(shore-2-miss =(- ?s2m 0))

(shore-1-cann =(+ ?s1c 2))

(shore-2-cann =(- ?s2c 2))

(boat-location shore-1)

(last-move =(move-string 0 2 shore-1)))

(duplicate ?node

(search-depth =(+ 1 ?num))

(parent ?node)

(shore-1-miss =(+ ?s1m 1))

(shore-2-miss =(- ?s2m 1))

(shore-1-cann =(+ ?s1c 0))

(shore-2-cann =(- ?s2c 0))

(boat-location shore-1)

(last-move =(move-string 1 0 shore-1)))

(duplicate ?node

(search-depth =(+ 1 ?num))

(parent ?node)

(shore-1-miss =(+ ?s1m 2))

(shore-2-miss =(- ?s2m 2))

(shore-1-cann =(+ ?s1c 0))

(shore-2-cann =(- ?s2c 0))

(boat-location shore-1)

(last-move =(move-string 2 0 shore-1)))

(duplicate ?node

(search-depth =(+ 1 ?num))

(parent ?node)

(shore-1-miss =(+ ?s1m 1))

(shore-2-miss =(- ?s2m 1))

(shore-1-cann =(+ ?s1c 1))

(shore-2-cann =(- ?s2c 1))

(boat-location shore-1)

(last-move =(move-string 1 1 shore-1))))

(defmodule CONSTRAINTS

(import MAIN deftemplate status))

(defrule CONSTRAINTS::cann-eats-miss1

(declare (auto-focus TRUE))

?node <- (status (shore-1-miss ?s1m)

(shore-1-cann ?s1c))

(test (and (> ?s1c ?s1m) (> ?s1m 0)))

=>

(retract ?node)

)

(defrule CONSTRAINTS::cann-eats-miss2

(declare (auto-focus TRUE))

?node <- (status (shore-2-miss ?s2m)

(shore-2-cann ?s2c))

(test (and (> ?s2c ?s2m) (> ?s2m 0)))

=>

(retract ?node)

)

(defrule CONSTRAINTS::stoping-illegal-move

(declare (auto-focus TRUE) )

?node <- (status (shore-2-miss ?s2m)

(shore-2-cann ?s2c)

(shore-1-miss ?s1m)

(shore-1-cann ?s1c))

(test (or (< ?s2m 0) (< ?s1m 0) (< ?s2c 0) (< ?s1c 0)))

=>

(retract ?node)

)

(defrule CONSTRAINTS::circular-path

(declare (auto-focus TRUE))

(status (search-depth ?sd1)

(shore-1-miss ?s1m)

(shore-1-cann ?s1c)

(shore-2-miss ?s2m)

(shore-2-cann ?s2c)

(boat-location ?shore))

?node <- (status (search-depth ?sd2&:(< ?sd1 ?sd2))

(shore-1-miss ?s1m)

(shore-1-cann ?s1c)

(shore-2-miss ?s2m)

(shore-2-cann ?s2c)

(boat-location ?shore))

=>

(retract ?node)

)

(defmodule SOLUTION

(import MAIN deftemplate status)

(import MAIN defglobal initial-missionaries)

(import MAIN defglobal initial-cannibals)

)

(deftemplate SOLUTION::moves

(slot id (type FACT-ADDRESS SYMBOL) (allowed-symbols no-parent))

(multislot moves-list (type STRING))

)

(defrule SOLUTION::goal-test

(declare (auto-focus TRUE))

?node <- (status (parent ?parent)

(shore-1-miss ?s1m)

(shore-1-cann ?s1c)

;(shore-2-miss ?\*initial-missionaries\*)

;(shore-2-cann ?\*initial-cannibals\*)

(last-move ?move)

)

(test (and (= ?s1c 0) (= ?s1m 0)))

=>

(retract ?node)

(assert (moves (id ?parent) (moves-list ?move)))

)

(defrule SOLUTION::build-solution

?node <- (status (parent ?parent)

(last-move ?move))

?mv <- (moves (id ?node) (moves-list $?rest))

=>

(modify ?mv (id ?parent) (moves-list ?move ?rest))

)

(defrule SOLUTION::print-solution

?mv <- (moves (id no-parent) (moves-list "No move" $?m))

=>

(retract ?mv)

(printout t t "Solution found: " t t)

(progn$ (?move ?m) (printout t ?move))

)

# Результат

