CONSTRAINT PROGRAMMING With Puzzles

Clojure & JSR-331

Oscar Riveros
(http://mx-clojure.blogspot.com/)

PROGRAMACION DE RESTRICCIONES Con Puzzles

Clojure & JSR-331

Oscar Riveros
(http://mx-clojure.blogspot.com/)

V002

February 23, 2013 5:29 PM

Introduction

CPWP is a set of problems of finite CONSTRAINT LOGIC PROGRAMMING of FINITE DOMAINS, in this document are specifically addressed in Clojure & JSR-331 API The Java Constraint Programming. Itself is a personal investigation, non-profit, is only shared to the public for what it is, a personal study of the issue being raised.

Introducción

CPWP es un conjunto de problemas de la CONSTRAINT LOGIC PROGRAMMING of FINITE DOMAINS, en este documento están específicamente resueltos en Clojure & JSR-331 The Java Constraint Programming API. En si es una investigación personal, sin fines de lucro, solamente es compartida al publico como lo que es, un estudio personal del tema que se plantea.

Basic function to solve the puzzles
Función básica para la solución de los puzzles

```
(ns cpwp.core
  (:import [javax.constraints
            Problem
            ProblemFactory
            Var]))

(defn solve
  [problem solution]
  (let [solver (.getSolver problem)
        solution-iterator (.solutionIterator solver)]
    (try

      (.log problem "Before Constraint Posting")
      (.log problem (.getVars problem))

      (solution)

      (.log problem "After Constraint Posting")
      (.log problem (.getVars problem))

      (.log problem "=== Find Solution:")

      (while (.hasNext solution-iterator)
        (.log (.next solution-iterator)))

      (.log problem "After Search")
      (.log problem (.getVars problem))

      (catch Exception exception
        (.log problem (.getMessage exception))))))

(defn solve-midi
  [problem solution make-music]
  (let [solver (.getSolver problem)
        solution-iterator (.solutionIterator solver)]
    (try

      (.log problem "Before Constraint Posting")
      (.log problem (.getVars problem))

      (solution)

      (.log problem "After Constraint Posting")
      (.log problem (.getVars problem))

      (.log problem "=== Find Solution:")

      (while (.hasNext solution-iterator)
        (let [next-solution (.next solution-iterator)]
          (make-music next-solution)
          (.log next-solution)))

      (.log problem "After Search")
      (.log problem (.getVars problem))

      (catch Exception exception
        (.log problem (.getMessage exception))))))
```

Arch Friends

Author: Mark T. Zegarelli
Publication: Dell Logic Puzzles
Issue: April, 1998
Page: 7
Stars: 1

Harriet, upon returning from the mall, is happily describing her four shoe purchases to her friend Aurora. Aurora just loves the four different kinds of shoes that Harriet bought (ecru espadrilles, fuchsia flats, purple pumps, and suede sandals), but Harriet can't recall at which different store (Foot Farm, Heels in a Handcart, The Shoe Palace, or Tootsies) she got each pair. Can you help these two figure out the order in which Harriet bought each pair of shoes, and where she bought each?

1. Harriet bought fuchsia flats at Heels in a Handcart.
2. The store she visited just after buying her purple pumps was not Tootsies.
3. The Foot Farm was Harriet's second stop.
4. Two stops after leaving The Shoe Place, Harriet bought her suede sandals.

Determine: Order - Shoes - Store

Arco de Amigos

Autor: Mark T. Zegarelli
Publicación: Dell Logic Puzzles
Edición: Abril de 1998
Página: *7*
Estrellas: 1

Harriet, a su regreso de un centro comercial, felizmente describe sus compras de de cuatro zapatos a su amiga Aurora. Aurora le encanta los cuatro diferentes tipos de zapatos que Harriet compro (alpargatas, tacones fucsia, zapatos púrpura y sandalias de gamuza), pero Harriet no recuerdo en qué diferente tienda (Foot Farm, Heels in a Handcart, The Shoe Palace, or Tootsies) obtuvo cada par. ¿Puedes ayudar a encontrar el orden en que Harriet compró cada par de zapatos, y donde ella compró cada uno?

1. Harriet compro tacones fucsia en Heels in a Handcart.
2. La tienda que visitó justo después de comprar sus zapatos púrpura no era Tootsies.
3. La Foot Farm era la segunda parada de Harriet.
4. Dos paradas después de abandonar The Shoe Palace, Harriet le compró unas sandalias de gamuza.

Determinar: Orden - Zapatos - Tienda

Solution - Solución

```
(ns cpwp.arch-friends
  (:use [cpwp.core])
  (:import [javax.constraints
            Problem
            ProblemFactory
            Var]))

(def problem (ProblemFactory/newProblem "Arch Friends"))

(defn solution
  []
  (let [flats    (.variable problem "flats"    1 4)
        espa     (.variable problem "espa"     1 4)
        pumps    (.variable problem "pumps"    1 4)
        sandals  (.variable problem "sandals"  1 4)
        foot     (.variable problem "foot"     1 4)
        heels    (.variable problem "heels"    1 4)
        shoe     (.variable problem "shoe"     1 4)
        tootsies (.variable problem "tootsies" 1 4)]
    (.postAllDifferent problem [flats espa pumps sandals])
    (.postAllDifferent problem [foot heels shoe tootsies])
    (.post problem foot "=" 2)
    (.post problem flats "=" heels)
    (.post problem (.plus pumps 1) "!=" tootsies)
    (.post problem (.plus shoe 2) "=" sandals)))

(solve problem solution)
```

Result - Resultado

```
JSR-331 "Constraint Programming API" Release 1.1.0
JSR-331 Implementation based on JSetL 2.3
Before Constraint Posting

After Constraint Posting
Var[0]: flats[1..4]
Var[1]: espa[1..4]
Var[2]: pumps[1..4]
Var[3]: sandals[1..4]
Var[4]: foot[1..4]
Var[5]: heels[1..4]
Var[6]: shoe[1..4]
Var[7]: tootsies[1..4]
=== Find Solution:
Solution #0:
       flats[4] espa[2] pumps[1] sandals[3] foot[2] heels[4] shoe[1] tootsies[3]
After Search
Var[0]: flats[4]
Var[1]: espa[2]
Var[2]: pumps[1]
Var[3]: sandals[3]
Var[4]: foot[2]
Var[5]: heels[4]
Var[6]: shoe[1]
Var[7]: tootsies[3]
```

Conclusion

First, purple pumps, The Shoe Place
Second, espadrilles, Foot Farm
Third, suede sandals, Tootsies
Fourth, fuchsia flats, Heels in a Handcart.

Conclusión

Primero, zapatos purpura, The Shoe Place
Segundo, alpargatas, Foot Farm
Tercero, sandalias de gamuza, Tootsies
Cuarto, tacones fucsia, Heels in a Handcart.

All Intervals Series

```
(An adaptation of All-Interval Series example of Strasheela by
Torsten Anders using Clojure, JSR-331 y JMusic)
```

This example defines an all-interval series (a classical CSP) and outputs the result in a number of different ways as a demonstration.

The example stems from dodecaphonic music composition. A series (or tone row) is a sequence of twelve tone names of the chromatic scale or twelve pitch classes, in which each pitch class occurs exactly once. In an all-interval series, also the eleven intervals between the twelve pitches are all pairwise distinct (i.e. each interval occurs only once). These intervals are computed in such a way that they are inversional equivalent: complementary intervals such a fifth upwards and a fourth downwards count as the same interval (namely *7*).

Todas las Series de Intervalos

```
(Una adaptación de All-Interval Series de Strasheela por Torsten
Anders, usando Clojure, JSR-331 y JMusic)
```

En este ejemplo se define una serie de todos los intervalo (a CSP clásico) y envía los resultados a ficheros midis, como una muestra.

El ejemplo se deriva de la composición de música dodecafónica. Una serie (o row del tono) es una secuencia de doce notas de la escala cromática o doce pitch clases, en el que cada nota de la escala ocurre exactamente una vez. En la serie de todos los intervalos, también los once intervalos entre las doce notas son todos distintos a pares (es decir, cada intervalo se produce sólo una vez). Estos intervalos se calculan de tal manera que son inversionalmente equivalentes: intervalos complementarios tal como la quinta hacia arriba y hacia abajo o una cuarta cuentan como el mismo intervalo (es decir *7*).

Solution - Solución

```
(ns cpwp.all-intervals-series
  (:use [cpwp.core])
  (:import [javax.constraints
            Problem
            ProblemFactory
            Var])
  (:import [jm.music.data
            Score
            Part
            Phrase
            Note])
  (:import [jm JMC])
  (:import [jm.util Write]))

(def problem (ProblemFactory/newProblem "All Intervals Series"))

(def l 12)

(defn make-list
  [prefix l min max]
  (let [?list '()]
  (for [i (range l)]
    (cond ?list (.variable problem (str prefix i) min max)))))

(defn solution
  []
  (let [xs  (make-list "xs"  l 0 (dec l))
        dxs (make-list "dxs" (dec l) 1 (dec l))]
    (.postAllDifferent problem xs)
    (.postAllDifferent problem dxs)

    (dotimes [i (dec l)]
      (when (> i 0)
        (.post problem (.mod (.plus (.minus (nth xs (inc i))
                                    (nth xs i)) l) l) "=" (nth dxs i))))

    (.post problem (first xs) "=" 0)
    (.post problem (last  xs) "=" (quot l 2))))


(defn make-music
  [next-solution]
  (let [score      (new Score "Mx Demo - All Intervals Series")
        piano-part (new Part "Piano" JMC/PIANO 0)
        phrase     (new Phrase "All Intervals Series" 0.0)
        notes      (map #(.getValue next-solution (str "xs" %)) (range (dec l)))]

    (doseq [note notes]
      (.addNote phrase (new Note (+ note 60) JMC/C))) ;(60 = c4)

    (.addPhrase piano-part phrase)
    (.addPart   score piano-part)

    (Write/midi score (str "/Google Drive/tmp/Mx/all-intervals-series"
                           next-solution ".mid"))))


(solve-midi problem solution make-music)
```

MIDIs

In the repository project, there is a zip file with 3856 midi files.
(https://github.com/maxtuno/Clojure---JSR-331---Puzzles)

MIDIs

En el repositorio del proyecto, hay un archivo zip con los 3856 archivos midi.
(https://github.com/maxtuno/Clojure---JSR-331---Puzzles)