Programación Lógica

Obligatorio 2 – 2012

Facultad de Ingeniería
Instituto de Computación
Grupo de Procesamiento de Lenguaje Natural

El objetivo de este obligatorio es resolver problemas de nivel intermedio y alto en Prolog y otras tecnologías, y analizar el desempeño de las soluciones.

Nota previa - IMPORTANTE

Se debe cumplir íntegramente el "Reglamento del Instituto de Computación ante Instancias de No Individualidad en los Laboratorios", disponible en `http://www.fing.edu.uy/inco/pm/uploads/Ense%f1anza/NoIndividualidad.pdf`

En particular está prohibido utilizar documentación de otros grupos, de otros años, de cualquier índole, o hacer público código a través de cualquier medio (news, correo, papeles sobre la mesa, etc.).

Forma de entrega

La entrega se realizará a través del espacio eva del curso, en una página destinada a tal fin que se habilitará cerca de la fecha de entrega. Se debe entregar un solo archivo '`grupo#.zip`', donde # es el número de grupo que realiza la entrega, conteniendo los ítems indicados en el apartado **Entregable**.

Fecha de entrega

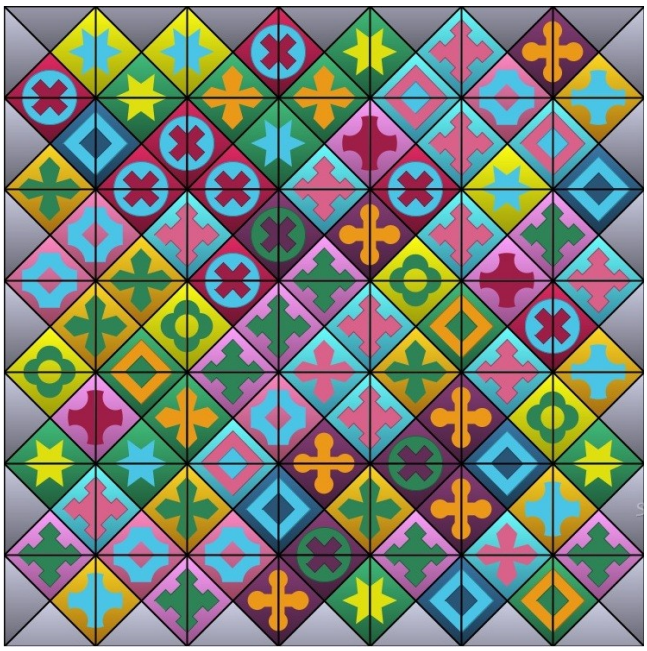
Los trabajos deberán ser entregados siguiendo el procedimiento descrito anteriormente antes del domingo 24 de junio de 2012 a las 23:30 horas, sin excepciones. No se aceptará ningún trabajo pasada la citada fecha.

Observaciones

La implementación debe realizarse de manera que pueda ser ejecutada en la plataforma SWI-Prolog, en sus versiones para Windows o Linux.

Resolvedor de puzzles

Los puzzles de correspondencia de bordes (edge-matching puzzles en inglés [1]) son un tipo de problemas ampliamente estudiados, para los cuales no se ha encontrado aún una solución eficiente. Un ejemplo famoso en el puzzle Eternity II [2], cuyos creadores ofrecían en 2007 un premio de dos millones de dólares a la primera persona que encontrara su solución.



Puzzle tipo Eternity II de 7x7

Un puzzle de edge-matching se caracteriza por tener un conjunto de piezas con bordes de diferentes colores y patrones, las cuales deben disponerse sobre un tablero de manera que los los bordes de las piezas adyacentes coincidan (nos concentraremos en este caso, aunque hay ejemplos de puzzles con otros tipos de restricciones). Cada pieza puede colocarse libremente en cualquier posición del tablero y rotarse de cualquier manera. Usualmente los bordes del tablero son de un color particular, lo que permite acotar algunas la posibilidad de combinaciones para algunas de las piezas.

El problema general de puzzles de edge-matching es NP-completo, por lo que es poco probable que encontremos una solución eficiente para todos los casos. Las formas de resolverlo siempre necesitan algún tipo de backtracking.

La combinación de colores y patrones que pueden tener los bordes se puede representar, sin pérdida de generalidad, mediante un valor entero. A este valor le llamamos simplemente “color”. Una pieza del puzzle se puede representar mediante una tupla de 4 colores, representando los cuatro colores de los bordes de la pieza.

Un puzzle queda determinado por la lista de piezas que utiliza. Para llenar un tablero de tamaño N necesitaremos N^2 piezas. Recordemos que cada pieza puede estar en cualquier lugar del tablero y rotada de cualquier forma, siempre que se respete que los colores leídos de manera horaria se correspondan con los de la 4-tupla que define la pieza.

La solución de un puzzle se compone de una lista de posiciones de piezas. Cada elemento de esta lista es una 4-tupla (x,y,p,r), donde:

- (x,y) corresponden a la fila y columna donde se ubica la pieza en el tablero solución.
- p es el número de pieza dentro de la lista de entrada, comenzando en 1
- r es un valor que indica la rotación de la pieza dentro de su posición, sus posibles

valores son:
- 0 si la pieza no está rotada
- 1 si la pieza está rotada 90º en sentido horario
- 2 si la pieza está rotada 180º en sentido horario
- 3 si la pieza está rotada 270º en sentido horario

Se pide:

1. Implementar en Prolog un resolvedor de puzzles de edge-matching. Este resolvedor debe tomar su entrada de un archivo en el formato indicado, y devolver su salida como la lista descripta anteriormente.
   Firma del predicado principal: **`solve_puzzle(+File,+N,?Solution)`**

2. Interesa tener un análisis de la eficiencia de la solución implementada. Para eso se debe implementar un metaintérprete en Prolog que ejecute un Goal y además devuelva una lista con todos los predicados invocados durante la ejecución, y cuántas veces se ejecutó cada uno. Los predicados deben estar en el formato estándar de Prolog: **`nombre/aridad`**
   Firma del predicado principal: **`profiler(+Goal,-Profile)`**

3. Realizar un análisis de la eficiencia del resolvedor de puzzles en cuanto a tiempo de ejecución y cantidad de invocaciones a predicados. Para eso se proveerán definiciones de puzzles de diferentes tamaños y cantidades de colores. Resulta de interés analizar el comportamiento de la solución en función tanto del tamaño del puzzle como de la cantidad de colores.
   - Para medir el tiempo de ejecución se utilizará el predicado time, ya definido en SWI-Prolog.
   - Para medir las cantidades de invocaciones se utilizará el metaintérprete generado en la parte anterior.

Cuadrado latino

En el obligatorio anterior implementamos en Prolog un predicado que nos permite obtener todos los cuadrados latinos de orden N. Pero el problema de encontrar una matriz cuyos valores cumplan con la restricción de un cuadrado latino también puede verse como un problema de satisfactibilidad booleana.

Existen sistemas especiales, llamados SAT-solvers, dedicados a resolver este tipo de problemas de manera eficiente. Resulta de interés compara la eficiencia de una solución implementada en Prolog con una solución obtenida utilizando un SAT-solver. Con este objetivo se utilizará un SAT-solver open source, de alta performance, denominado MiniSat [3].

Utilizando este sistema se debe construir en Prolog un programa cuyo predicado principal tenga la siguiente firma:

```
latin_square_sat(+N, ?M)
```

Donde N es el orden del cuadrado latino a construir, y M es la variable de salida donde se obtendrá la matriz que representa dicho cuadrado.
Este predicado debe realizar lo siguiente:

- Construir en un archivo **`latin.in`** la especificación del problema SAT para encontrar el cuadrado latino, en el formato establecido por MiniSat.
- Invocar el sistema MiniSat para que tome como entrada el archivo construido, y guarde su salida en otro archivo **`latin.out`**.
- Leer el resultado devuelto por MiniSat y construir la matriz correspondiente al cuadrado latino para devolver en la variable M.

Con esto obtendríamos un cuadrado latino calculado mediante SAT. Sin embargo este algoritmo no es comparable con nuestro algoritmo en Prolog, que nos permite encontrar todos los cuadrados latinos del mismo orden. Para lograr esto en SAT, podemos utilizar

el siguiente procedimiento. Supongamos que Q es el conjunto de clásusulas de la codificación original.

- Invocar SAT con las cláusulas Q: devuelve una solución S1
- Construir a partir de S1 una cláusula nS1, que represente la restricción de que el resultado S1 no es válido.
- Invocar SAT con las cláusulas Q ^ nS1: devuelve una solución S2 (que forzosamente será distinta de S1)
- Construir a partir de S2 una cláusula nS2, que represente la restricción de que el resultado S2 no es válido.
- Invocar SAT con las cláusulas Q ^ nS1 ^ nS2: devuelve una solución S3 (que será diferente de S1 y S2)
- **...**

De esta manera se pueden obtener todas las soluciones que se deseen. Sin embargo, recordemos que el número de cuadrados latinos de orden N crece muchísimo con N (para N=7 hay en el orden de 10^7 cuadrados). Por lo tanto no es factible esperar a obtener todos los cuadrados posibles. La idea es tener un procedimiento que nos devuelva los primeros K cuadrados distintos de orden N.

Se pide:

a) Prolog y SAT-solver

1. Especificar el esquema de codificación del problema SAT para encontrar un cuadrado latino de orden N.
2. Implementar en Prolog un programa que busca cuadrados latinos utilizando MiniSat, siguiendo el esquema propuesto.
   Firma del predicado principal: **`latin_square_sat(+N, ?M)`**
3. Implementar en Prolog un programa que retorna los K primeros cuadrados latinos de orden N, buscándolos mediante la solución del problema SAT.
   Firma del predicado principal: **`latin_square_sat_many(+N,+HowMany,?Bag)`**

b) Prolog

Implementar en Prolog un programa que realice un procesamiento similar a findall, pero retorne los primeros K resultados del predicado en vez de todos los posibles resultados. El predicado no debe recorrer todos los resultados posibles, sino que debe terminar su ejecución después de encontrar el K-ésimo.
Firma del predicado principal: **`n_values(+Goal,+Template,+K,-Bag)`** donde Bag es una lista con los K primeros resultados.

Utilizaremos este predicado para obtener los K primeros cuadrados latinos de orden N mediante la invocación:
`n_values(latin_square(N,M),M,K,Bag)`

c) Comparación

Realizar una comparación de la eficiencia de las dos implementaciones para obtener cuadrados latinos (a través de SAT, y usando Prolog puro) en cuanto a tiempo de ejecución en función del orden del cuadrado N.
Interesa distinguir, para el caso de SAT, qué proporción del tiempo se encuentra codificando la entrada, ejecutando el SAT-solver, y decodificando la entrada.

Entregable

El archivo a entregar debe contener:

1. Documentación que incluya:
   a) Especificación del esquema de codificación SAT.
   b) Análisis de performance solicitados.
   c) Archivos que componen la solución.
   d) Descripción de la solución implementada incluyendo las decisiones de diseño relevantes.
2. Implementación de todos los predicados Prolog requeridos.

Insumos

Se proveerá:

- Archivos Prolog con las firmas de los predicados para poder utilizarse como templates.
- Ejemplos de especificaciones de puzzles para diferentes tamaños y cantidades de colores.
- Binarios de MiniSat compilados para ejecutar en la plataforma Windows.

Nota

Es necesario que en el informe figuren el nombre y la cédula de identidad de cada integrante del grupo. En caso que esto no se cumpla el obligatorio no será corregido, con la consecuente pérdida del curso de sus autores. Solo se aceptarán informes en formato PDF (ver http://www.universidad.edu.uy/odfpdf/).

Referencias

[1] Edge-matching puzzle - http://en.wikipedia.org/wiki/Edge-matching_puzzle
[2] Eternity 2 - http://en.wikipedia.org/wiki/Eternity_II_puzzle
[3] MiniSat - http://minisat.se/