Resolução do puzzle Yin Yang usando Programação em Lógica com Restrições

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Yin Yang_4

Filipe Coelho – 201500072

Descrição do processo de implementação de uma possível solução para o puzzle *Yin Yang*. Nesta implementação foi utilizado prolog com restrições, onde o domínio das variáveis foi limitado e as regras de posicionamento foram definidas antes serem instanciadas.

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

23 de Dezembro de 2016

Conteúdo

Introdução

Neste trabalho foi pedido para implementar uma solução para um determinado puzzle usando programação em lógica com restrições. Um dos pontos importantes é a necessidade das variáveis e o seu domínio terem regras definidas antes de serem instanciadas automaticamente pelo software. Neste caso específico foi escolhido o puzzle Yin Yang, um puzzle de relativa fácil análise mas de uma implementação mais exigente. O grande objetivo deste trabalho consiste em aprender como é possível calcular soluções para um problema recorrendo apenas a restrições no domínio.

Descrição

O puzzle Yin Yang, escolhido para a realiação deste trabalho, tem como base um tabuleiro quadrangular em que todas as casas têm de estar preenchidas com um círculo preto ou branco, mas com duas condições: Todos os círculos da mesma cor devem de estar ligados, ou horizontalmente ou verticalmente (ou ambos); Um grupo de 2x2 casas não pode conter círculos da mesma cor.

Com base nessas regras e com a ajudar do sistema de desenvolvimento SICStus Prolog, foram pensadas em restrições a serem implementadas na liguagem prolog para tentar satisfazer estas condições enunciadas. Para atingir este resultado recorri a conteúdos apresentados nas aulas teóricas e teorico-práticas da disciplina de programação, manuais de prolog e alguma pesquisa na internet, onde uma das mais relevantes foi o blog[1] de Maharashtra Mumbai onde ele apresenta estratégias e métodos para a resolução para este e outros puzzles.

[1] http://rohanrao.blogspot.pt/

Abordagem

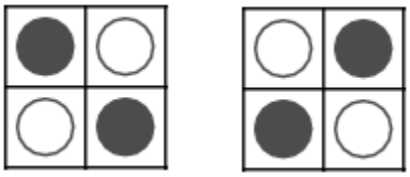
Variáveis de decisão

Como variáveis de decisão foi utilizada uma lista de tamanho variável onde foram introduzidas variáveis não inicializadas. Como domínio foi apenas utilizado os números 0 e 1, sendo que num outro predicado onde é feita a representação do tabuleiro, os 0's foram traduzidos para espaços em branco e os 1's para '*', simbolizando um círculo preto.

Restrições

Uma das restrições que, com base na análise do puzzle, salta logo à vista: um quadrado de 2x2 não pode ser todo da mesma cor. Isso foi das primeiras restrições implementadas no código.

A seguinte foi derivada da conectividade das cores: num quadrado 2x2, as diagonais não podem ter a mesma cor entre si e serem diferentes uma da outra, ou seja, isto não pode acontecer:



Uma terceira restrição, também derivada da conectividade das cores foi que não poderia haver mais de duas alterações de cor no nível mais externo do tabuleiro, caso contrário haveriam peças que ficariam isoladas. Isso traduziu-se numa restrição que unia a linha superior e inferior com a coluna mais à esquerda e mais à direita, por ordem, e verifica, através de materialização, se existe mais de duas mudanças.

Uma última restrição criada foi a de, para um tabuleiro com lado maior do que 3, não poder haver uma peça onde à sua volta só existam peças da cor contrária. Esta restrição era para ser expandida para grupos de peças, formados de forma automática, mas não foi possível devido ao já clarificado na introdução.

Função de Avaliação

Para avaliar o resultado obtido foram criados predicados de conversão da lista para tabela e posteriormente mostrado ao utilizador. Através da análise do gráfico era possível ver se o problema tinha sido corretamente solucionado ou não.

Estratégia de Pesquisa

Como estratégia de pesquisa foi utilizada a opção *down* no labeling, porque trouxe resultados muito mais fiáveis e próximos do pretendido na pesquisa de elementos.

Visualização da Solução

Para representar o tabuleiro na consola foram utilizados vários predicados:

displayRow(RowSize, Rows, Vars) - este é o predicado que trata da representação das diversas linhas, e é chamado para cada linha que exista na tabela. Este predicado chama outros importantes para a representação, como o displayHeader e o displayRowContent, bem como a si próprio para a próxima linha ser desenhada;

displayHeader(N) – este predicado trata de representar as divisões entre as várias linhas, sendo N o número de colunas que exista no tabuleiro, a fim da representação se tornar escalável com o problema;

displayRowContent(Count, Index, Vars) – este predicado é o que desenha cada célula. Ele é chamado para o cada coluna que exista na tabela, e trata de desenhar as divisões entre as células. Faz uso de um outro predicado, displayCell para representar o conteúdo da célula;

displayCell(1)/ displayCell(_) – este predicado representa a peça em si: quando o valor 0 está na célula correspondente, ele desenha um espaço em branco. Quando é o valor 1 é desenhado um asterisco.

Resultados

Para obter a representação do tabuleiro basta chamar o predicado yinyang(N), em que N é o número de células que o tabuleiro tem por linha/coluna. Ou seja, yinyang(2) traduz-se num tabuleiro com 4 casas, 2 linhas e 2 colunas:

```
| ?- yinyang(2).
+-+-+
|*|*|
+-+-+
|*| |
+-+-+
```

Para N igual a 2 e N igual a 3 o programa apresenta soluções corretas. No entanto para N maior do que 3 alguns casos falham, embora esteja bastante perto do resultado pretendido.

```
| ?- yinyang(3).
+-+-+-+
|*|*|*|
+-+-+-+
|*|  |*|
+-+-+-+
|*|  |*|
+-+-+-+
```

```
| ?- yinyang(6).
+-+-+-+-+-+-+
|*|*|*|*|*|*|
+-+-+-+-+-+-+
|*|  |*|  |*| |
+-+-+-+-+-+-+
|*|  |*|  |*| |
+-+-+-+-+-+-+
|*|*|*|*|*| |
+-+-+-+-+-+-+
|*|  |*|  |*| |
+-+-+-+-+-+-+
|*|  |  |  |  |
+-+-+-+-+-+-+
```

Em qualquer um dos casos até N menor ou igual a 6 a apresentação do resultado é praticamente instantânea. Para N igual a 7 já é necessário esperar alguns segundos (cerca de 10 num sistema relativamente recente), e para N igual a 8 o tempo de espera já aumenta bastante. Os casos que falham estão ligados com a continuidade das peças

da mesma cor e eventualmente poderiam ser resolvidos expandindo a restrição que está aplicada a uma única peça para grupos de peças.

Conclusões e Trabalho Futuro

Com este trabalho foi óbvia a capacidade de tecnologias como o prolog ajudarem na resolução de problemas onde apenas são conhecidos algumas restrições e pretende-se obter a melhor solução, e a sua utilidade para problemas de optimização não passa de todo despercebida. No entanto a sua divergência de linguagens compiladas e a sua sintaxe tornam-no numa ferramenta um pouco difícil de aprender e trabalhar.

Relativamente à programação com restrições não é algo que estivesse familiarizado, tendo sido espantado com a capacidade de resolução de problemas dando apenas algumas regras básicas.

No caso concreto deste trabalho, fica a pena de não estar 100% implementado, infelizmente a desistência de um membro do grupo em cima do prazo de entrega fez com que não fosse possível para mim implementar tudo o que supostamente seria feito pelo colega do grupo (como é o caso da verificação da continuidade de peças da mesma cor). No entanto penso que, e falo por mim, os conhecimentos foram adquiridos e compreendidos.

Como trabalho futuro gostaria de acabar a implementação das restrições, bem como a possíbilidade de fornecer uma tabela incompleta ao programa e ele ser capaz de descobrir uma solução.

Biografia

[1] – Documentos fornecidos pelos professores da cadeira de programação em lógica da faculdade de engenharia da universidade do Porto

[2] – Documento descritivo do puzzle escolhido, fornecido pelos professores da cadeira

[3] – Manual de referência do SWI Prolog - http://www.swi-prolog.org/pldoc/doc_for?object=manual

[4] – Blog de Maharashtra Mumbai- http://rohanrao.blogspot.pt/