Dominó

Resolução de Problema de Decisão utilizando Progamação em Lógica com Restrições

Daniela Sá - up201405457
Tiago Silva - up201402841

December 23, 2016

Resumo

Este trabalho foi realizado no contexto da unidade curricular de Programação em Lógica. Sendo assim, o objetivo do trabalho era criar um programa em Prolog com restrições que conseguisse resolver o puzzle de dominó. A solução para este problema faz uso de muitas das noções e predicados de Prolog que aprendemos nas aulas teóricas e práticas na parte final do semestre.

Keywords: sicstus, prolog, domino, feup

1 Introdução

O projeto teve como objetivo familiarizar-nos com programação em lógica com restrições através da implementação da solução de um puzzle de lógica usando restrições em prolog.
Já este artigo visa uma ponderação por parte dos alunos sobre o trabalho realizado, para que possam interiorizar melhor a teoria por trás do código que escreveram.
Sendo assim, o artigo está dividido em seis secções:

1 - introdução ao contexto do trabalho
2 - descrição do problema escolhido
3 - abordagem usada na resolução
4 - visualização da solução
5 - resultados das avaliação das opções de *labeling*
6 - conclusões
7 - referências usadas na resolução do problema

2 Descrição do Problema

O tema escolhido pelo nosso grupo foi "Dominos". Neste problema é fornecido um tabuleiro já preenchido com dominós sem divisórias, assim como uma lista das peças colocadas. Esta lista contém todas as combinações de metades de peça um certo intervalo.
Então, o objetivo do puzzle é encontrar a combinação correta de divisórias para que as peças no tabuleiro correspondam exatamente às peças na lista.

1	3	0	1	2
3	2	0	1	3
3	3	0	0	1
2	2	1	2	0

Figure 1: Tabuleiro inicial

1	3	0	1	2
3	2	0	1	3
3	3	0	0	1
2	2	1	2	0

Figure 2: Tabuleiro resolvido

3 Abordagem

3.1 Variáveis de Decisão

Para representar o tabuleiro de jogo onde vamos aplicar as restrições, usamos uma única lista preenchida com variáveis a que demos significados pertinentes:

A – cantos do tabuleiro
Sxy – fronteira sul do quadrado (x,y)
Exy – fronteira este do quadrado (x,y)
2 a 8 – cada número representa o número de pontos em cada metade de dominó
11 – representa uma metade de dominó de 0 pontos
12 – representa uma metade de dominó de 1 ponto

```
Solution = [ A,  S01, A, S02, A, S03, A, S04, A, S05,  A,
             E10,  8, E11, 3, E12, 7, E13, 8, E14, 2,  E15,
              A,  S11, A, S12, A, S13, A, S14, A, S15,  A,
             E20,  3, E21, 2, E22, 7, E23, 8, E24, 3,  E25,
              A,  S21, A, S22, A, S23, A, S24, A, S25,  A,
             E30,  3, E31, 3, E32, 7, E33, 7, E34, 8,  E35,
              A,  S31, A, S32, A, S33, A, S34, A, S35,  A,
             E40,  2, E41, 2, E42, 8, E43, 2, E44, 7,  E45,
              A,  S41, A, S42, A, S43, A, S44, A, S45,  A   ],
```

Figure 3: Exemplo de lista onde aplicamos restrições

Enquanto que para restringir o domínio das variáveis utilizamos outra lista, também representante do tabuleiro.
O domínio das variáveis nesta lista é $[0, 1]$, onde 0 é uma fronteira exterior e 1 uma linha interior. Nesta lista o significado das variáveis é similar ao da lista utilizada para as restrições:

Sxy – fronteira sul do quadrado (x,y)

Exy – fronteira este do quadrado (x,y)

```
Borders = [ S01, S02, S03, S04, S05,
            E10, E11, E12, E13, E14, E15,
            S11, S12, S13, S14, S15,
            E20, E21, E22, E23, E24, E25,
            S21, S22, S23, S24, S25,
            E30, E31, E32, E33, E34, E35,
            S31, S32, S33, S34, S35,
            E40, E41, E42, E43, E44, E45,
            S41, S42, S43, S44, S45       ],
```

Figure 4: Exemplo de lista onde restringimos domínios

3.2 Restrições

Usamos apenas duas restrições no programa:

- "neighbourship contraint": garante que as direções das peças são consistentes. Isto é, que nos arredores de uma metade de dominó haverá apenas uma divisória a conectá-la com outra metade. Caso contrário, teríamos peças de com mais de duas metades ligadas.
  Para concretizar isto, usamos uma lista para restrições como a ilustrada na figura 3. Vamos à posição de cada metade de peça e verificamos que a soma das divisórias que a rodeiam é igual a 1.
  Por exemplo, para a metade (2,4), a restrição é S14 + E23 + S24 + E24 = 1.

- "placement constraint": consiste em garantir que cada dominó está colocado uma só vez no tabuleiro. Para isto temos que encontrar todas as colocações possíveis de cada dominó e verificar que a soma das fronteiras entre as suas metades resulta em 1.
  Por exemplo, o dominó <0,2> tem três posições possíveis na figura 1. Neste caso a restrição é E22 + S34 + E44 = 1, pois apenas uma dessas fronteiras pode efetivamente unir as metades <0> e <2>.

3.3 Função de Avaliação

Não foi necessário implementar uma função de avaliação pois verificámos as soluções em formato de texto, quando são imprimidas.

3.4 Estratégia de Pesquisa

Em termos de ordenação de variáveis e seleção de valores usamos as opções *ffc* e *bisect*, respetivamente. Como referido nas aulas teóricas, são as duas opções que produzem melhores resultados em termos de eficiência do labeling.

4 Visualização da Solução

Para representar o tabuleiro em modo de texto fizemos uso da lista usada no labeling, que transformámos em matriz, percorrendo as linhas e mapeando cada caracter da seguinte forma:
Sendo assim, imprimimos as soluções neste formato:

```
toChar(0, '-', sb).
toChar(0, '|', sq).
toChar(1, ':', _).
toChar(9, ' ', _).
toChar(11, 0, _).
toChar(12, 1, _).
toChar(10, ' ', _).
toChar(X, X, _).
```

Figure 5: Mapeamento dos elementos da matriz solução para impressão

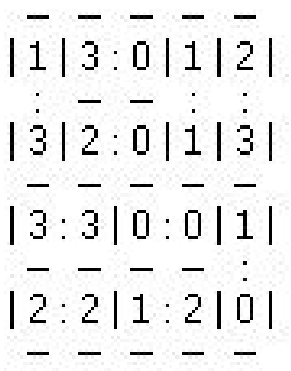

Figure 6: Exemplo de visualização

5 Resultados

Para decidir que opções colocar no *labeling* do nosso programa fizemos testes com *ffc* e *bisect* em dois tabuleiros de complexidade diferente. Obtivemos os seguintes resultados, impressos na consola do SICStus:

```
Elapsed time: 0.009
Resumptions: 1146
Entailments: 461
Prunings: 1350
Backtracks: 3
Constraints created: 390
```

```
Elapsed time: 0.008
Resumptions: 1146
Entailments: 461
Prunings: 1350
Backtracks: 3
Constraints created: 390
```

```
Elapsed time: 0.009
Resumptions: 1146
Entailments: 461
Prunings: 1404
Backtracks: 3
Constraints created: 390
```

```
Elapsed time: 0.010
Resumptions: 1146
Entailments: 461
Prunings: 1404
Backtracks: 3
Constraints created: 390
```

Figure 7: Estatísticas num tabuleiro simples. Da esquerda para a direita: *labeling* vazio, com *ffc*, com *bisect*, e ambos.

```
Elapsed time: 0.026
Resumptions: 1826
Entailments: 739
Prunings: 2085
Backtracks: 2
Constraints created: 638
```

```
Elapsed time: 0.026
Resumptions: 1826
Entailments: 739
Prunings: 2085
Backtracks: 2
Constraints created: 638
```

```
Elapsed time: 0.026
Resumptions: 1826
Entailments: 739
Prunings: 2154
Backtracks: 2
Constraints created: 638
```

```
Elapsed time: 0.026
Resumptions: 1826
Entailments: 739
Prunings: 2154
Backtracks: 2
Constraints created: 638
```

Figure 8: Estatísticas num tabuleiro complexo. Da esquerda para a direita: *labeling* vazio, com *ffc*, com *bisect*, e ambos.

Observamos, então, que *ffc* é a melhor escolha para o nosso trabalho. Porque apesar de num tabuleiro complexo não fazer diferença, num tabuleiro simples o desempenho melhora um pouco. Decidimos também não usar a opção *bisect*, pois piorava ou mantinha a eficácia do *labeling*.

6 Conclusões

Este trabalho visava o desenvolvimento de um programa em prolog com restrições que solucionasse o puzzle dominó. Com isto, ganhamos conhecimentos sobre programação em lógica com restrições em domínios finitos.
O programa está implementado em sete ficheiros, mas basta consultar main.pl no SICStus. Para iniciar o programa deve introduzir-se o comando *main.* .
Ao realizar este projeto compreendemos a um nível mais intrínseco a teoria aprendida durante a parte final do semestre, o que nos permitirá fazer melhor proveito da unidade curricular em geral.

7 Referências

Para nos auxiliar na construção da solução ao puzzle de dominós usamos o seguinte artigo:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.97.9715&rep=rep1&type=pdf