Resolução e Elaboração de jogos KenKen implementado em Prolog

Diogo Pinto ei11120 and Wilson Oliveira ei11085

FEUP-PLOG, Turma 3MIEIC06, Grupo 67
www.fe.up.pt

Abstract. O objectivo do trabalho é o desenvolvimento de um programa em prolog com a capacidade de resolver problemas do conhecido jogo de puzzle numerico *KenKen*, e também a elaboração de puzzles deste jogo. Para a resolução deste problema será utilizado técnicas de programação com restrições em prolog.

Keywords: KenKen, Prolog, Restrições, Resolução, Geração

1 Introdução

Este trabalho tinha como objectivo a utilização de programação com restrições em prolog, para a resolução de problemas de optimização ou decisão, verificando assim a eficácia deste método de programação em relação a métodos de tentativa e erro. Para a realização deste objectivo decidimos implementar um programa capaz de resolver puzzle numéricos do conhecido jogo KenKen. Neste artigo é possivel analisar os métodos utilizados para a resolução do problema em prolog, assim como excertos mais importantes do código implementado.

2 Descrição do problema

A implementação de um programa capaz de resolver puzzles numéricos do tipico jogo KenKen, um jogo similar a Sudoku, em que temos um tabuleiro NxN com números diferentes em cada linha e coluna. Este tabuleiro é subdividido em campos, para os quais é indicado uma operação matemática e um resultado. Estes campos impôem restricões no preenchimento do tabuleiro, uma vez que os números utilizados em cada campo, após a aplicação da operação matemática indicada, devem originar o resultado pedido.

Para a resolução deste problema foram utilizados métodos de programação com restricões em prolog, impondo restrições nos numeros que podem ser utilizados, verificando que não existem números repetidos por linha e coluna, e verificando que as operações matemáticas indicadas se evidenciam.

3 Ficheiros de Dados

4 Variáveis de Decisão

Para a resolução deste problema são necessárias quatro variáveis de decisão, primeiro o dominio dos números a serem utilizados no tabuleiro. Só podem ser utilizados no tabuleiro números de valor igual ou inferior ao tamanho do tabuleiro, por exemplo, se o tabuleiro for 4x4, só poderão ser utilizados números de um a quatro. A segunda e terceira restrição são similares, deve ser verificado que não existem números repetidos, por coluna e linha. A última restrição deve verificar que as operações matemáticas pedidas e respectivo resultado se verificam. Fazendo a verificação destas quatro restrições estaremos a reduzir eficazmente o dominio de pesquisa de possiveis soluções.

5 Restrições

A implementação da restrição do dominio, é feita criando uma lista com as variáveis de dominio que irão ser preenchidas, e impondo a restrição do valor máximo a ser utilizado ($SupLim$).

```
imposeDomainConstrainInRow([cell(_, Value) | Rs], SupLim) :-
Value in 1..SupLim,    imposeDomainConstrainInRow(Rs, SupLim).
```

A implementação da restrição de números diferentes numa linha é feita através da construção de uma lista com as variáveis de dominio, e utilizando a função .....", para adicionar esta restrição. A restrição para as colunas é feita utilizando o mesmo predicado, mas é utilizado a função *transpose* para inverter a ordem das listas que formam o tabuleiro.

```
imposeRowConstrain([], List) :- all_distinct(List).

imposeRowConstrain([cell(_, Value) | Rs], List) :-
append([Value], List, NewList),
imposeRowConstrain(Rs, NewList).
```

A implementação da restrição das operações matemáticas e feita construindo uma lista com as celulas que compoem um campo, e mediante a operação especificada nesse campo, é imposta a restrição.

```
imposeFieldConstrain(Board, [field(FID, Op, Res) | Fs]) :-
getFieldCells(FID, Board, L),
applyOpConstrain(L, Op, Res),
imposeFieldConstrain(Board, Fs).
```

6 Função de avaliação

Penso que nao se aplica no nosso trabalho

7 Estratégia de Pesquisa

Na resolução de um tabuleiro, após a imposição das restrições é executado o *labelling* com os valores por defeito, para obter a melhor solução possivel para o tabuleiro apresentado.

Na geração de um tabuleiro, após a imposição das restrições nas variáveis de dominio, o labelling é feito um número aleatório de vezes, com o objectivo de obter sempre tabuleiros diferentes e válidos para o mesmo tamanho pedido.

8 Visualização da solução

Os predicados de visualização são chamados automaticamente após a resolução de um tabuleiro. Os predicados de impressão são bastante simples, fazendo uso da recursividade de prolog e de iteradores para fazer a construção do tabuleiro, e numerar as colunas e linhas, para facilitar a identificação dos campos internos do tabuleiro. Como os campos são aleatórios, fizemos uso de carateres ascii para salientar a diferença entre campos e ao mesmo tempo tornar o tabuleiro mais

apelativo. Também é impressa um lista com os detalhes dos campos (operação matemática, e resultado final) e fazendo uso da numeração do tabuleiro, a coluna e linha onde se inicia o campo, sendo facilmente identificável as restantes células desse campo.

Predicado responsável por salientar a mudança de campo no tabuleiro, entre células adjacentes na mesma linha.

```
printRow([C1]) :-
cell(_, Value1) = C1,
printNumber(Value1),
write('||\n').

printRow([C1, C2 | Cs]) :-
cell(FieldID1, Value1) = C1,
cell(FieldID2, _Value2) = C2,
FieldID1 = FieldID2,
printNumber(Value1),
write('|'),
printRow([C2 | Cs]).

printRow([C1 | Cs]) :-
cell(_, Value1) = C1,
printNumber(Value1),
write('||'),
printRow(Cs).
```

Predicado responsável por salientar a mudança de campo no tabuleiro, entre células adjacentes na mesma coluna.

```
printHorizMidBorder([R1 | R1s], [R2 | R2s]) :-
cell(FieldID1, _) = R1,
cell(FieldID2, _) = R2,
FieldID1 = FieldID2,
write('---='),
printHorizMidBorder(R1s, R2s).

printHorizMidBorder([_R1 | R1s], [_R2 | R2s]) :-
write('===='),
printHorizMidBorder(R1s, R2s).
```

9 Resultados

10 Conclusões e perspectivas de desenvolvimento

Após análise dos resultados obtidos e diferentes testes, podemos concluir que os métodos de programação com restrições se tornam mais eficazes que métodos de

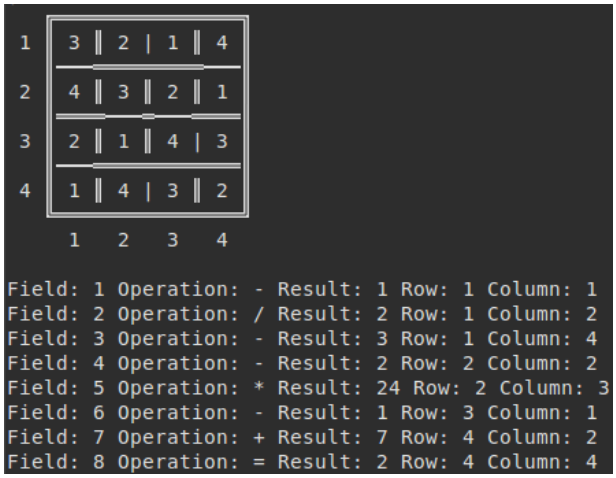

Fig. 1. Picture of a solved board.

tentativa e erro, uma vez que limitam à priori o nível de possibilidades para a resolução do problema tornando mais eficaz a procura de uma solução válida.

11 Bibliografia

References

1. Sicstus Documentation, `http://sicstus.sics.se/sicstus/docs/latest/html/sicstus.html/`
2. Swi-Prolog Documentation, `http://www.swi-prolog.org/pldoc/refman/`

A kenken.pl