

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 possível 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 típico jogo KenKen, um jogo similar a Sudoku, em que temos um tabuleiro $N \times N$ 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 impõem restrições 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 restrições em prolog, impondo restrições nos números que podem ser utilizados, verificando que não existem números repetidos por linha e coluna, e verificando que as operações matemáticas indicadas se evidenciam.

3 Variáveis de Decisão

Para a resolução deste problema são necessárias quatro variáveis de decisão, primeiro o domínio 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 4×4 , 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 domínio de pesquisa de possíveis soluções.

4 Restrições

A implementação da restrição do domínio, é feita criando uma lista com as variáveis de domínio 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 domínio, 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 é feita construindo uma lista com as células 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).
```

5 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 possível para o tabuleiro apresentado.

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

6 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 caracteres ascii para salientar a diferença entre campos e ao mesmo tempo tornar o tabuleiro mais apelativo. Também é impressa uma 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).

```

1	3		2		1		4
2	4		3		2		1
3	2		1		4		3
4	1		4		3		2
	1		2		3		4

Field: 1 Operation: - Result: 1 Row: 1 Column: 1
Field: 2 Operation: / Result: 2 Row: 1 Column: 2
Field: 3 Operation: - Result: 3 Row: 1 Column: 4
Field: 4 Operation: - Result: 2 Row: 2 Column: 2
Field: 5 Operation: * Result: 24 Row: 2 Column: 3
Field: 6 Operation: - Result: 1 Row: 3 Column: 1
Field: 7 Operation: + Result: 7 Row: 4 Column: 2
Field: 8 Operation: = Result: 2 Row: 4 Column: 4

Fig. 1. Picture of a solved board.

7 Resultados

Após a realização de vários testes, com geração e resolução de tabuleiros KenKen, o programa demorou sempre menos de 1 segundo a gerar e resolver o tabuleiro. Foi constatado para além disso que para tabuleiros com lado maior ou igual que dez, o tempo de resolução aumenta consideravelmente, devido a complexidade do puzzle.

Board Size	Creating Time	Solving Time
4	0.010	0.000
6	0.000	0.000
8	0.000	0.000
10	0.010	222.720

Fig. 2. Results of different tests.

8 Conclusões e perspectivas de desenvolvimento

Para terminar, é importante salientar que foi necessário a reestruturação do programa, uma vez que a variada complexidade dos puzzles levava a quebra do sistema. Após análise dos resultados obtidos e diferentes testes, podemos concluir que os métodos de programação com restrições se tornam mais eficazes que métodos de 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.

9 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

```

1      :- use_module(library(clpfd)).
2      :- use_module(library(lists)).
3      :- use_module(library(random)).
4      :- use_module(library(between)).
5      :- use_module(library(system)).
6
7      testBoard([[cell(1, -), cell(2, -), cell(2, -), cell(3, -
8          )],
9                  [cell(1, -), cell(4, -), cell(5, -),
10                     cell(3, -)],
11                  [cell(6, -), cell(4, -), cell(5, -),
12                     cell(5, -)],
13                  [cell(6, -), cell(7, -), cell(7, -),
14                     cell(8, -)])].
15
16      testBoardPrint([[cell(1, 5), cell(2, 5), cell(2, 555),
17          cell(3, 5)],
18                      [cell(1, 55), cell(4, 5), cell(5, 5),
19                         cell(3, 5)],
20                      [cell(6, 5), cell(4, 55), cell(5, 5),
21                         cell(5, 5)],
22                      [cell(6, 5), cell(7, 5), cell(7, 5),
23                         cell(8, 55)])].
24
25      testFields([field(1, '-', 1),
26                  field(2, '/', 2),
27                  field(3, '-', 3),
28                  field(4, '-', 2),
29                  field(5, '*', 24),
30                  field(6, '-', 1),
31                  field(7, '+', 7),
32                  field(8, '=', 2)]).
33
34      % for testing the domain constrains
35      testBoard2([[cell(1, -), cell(1, -)],
36                  [cell(1, -), cell(1, -)])].
37
38      testFields2([field(1, +, 6)]).
39
40      cell(_FieldID, _Value).
41      field(_FieldID, _Op, _FinalValue).
42
43      solveBoard :- testBoard(Board),

```

36	testFields(Fields),	
37	length (Board, Size),	
38	imposeDomainConstrain(Board,	
	Size),	
39	imposeRowConstrain(Board),	
40	imposeColumnConstrain(Board),	
41	imposeFieldConstrain(Board,	
	Fields),	
42	getValsList(Board, List),	
43	labeling([], List),	
44	printBoard(Board),	
45	write ('\\n\\n'),	
46	printFieldTable(Board, Fields).	
47		
48	solveBoard(Board, Fields) :- statistics (runtime, [T0, -])	
49	,	length (Board , Size) ,
50		imposeDomainConstrain (Board , Size) ,
51		imposeRowConstrain (Board) ,
52		imposeColumnConstrain (Board)

53

```
,  
imposeFieldConstrain  
(  
    Board  
,  
    Fields  
)  
,
```

54

```
getValsList  
(  
    Board  
,  
    List  
)  
,
```

55

```
labeling  
([ ],  
    List  
)  
,
```

56

```
statistics  
(  
    runtime  
,  
    [  
        T1  
,  
        -  
    ])  
,
```

57

```
printBoard  
(  
    Board  
)
```

58		, write (, \ n \ n ,) ,
59		printFieldTable (Board , Fields) ,
60		T is T1 — T0 ,
61		format (, ~ n ~ nSolving ~ took ~ ~3 d

```

~
sec
.~
n
,
,
[
T
])
,
62 fd_statistics
.
63
64
65 createBoard(Size) :- statistics(runtime, [T0, _]),
66                          length(Board,
67                               Size),
68                          now(Now),
69                          setrand(Now),
70                          initBoard(Size,
71                               Board),
72                          imposeDomainConstrain
73                          (Board, Size)
74                          ,
75                          imposeRowConstrain
76                          (Board),
77                          imposeColumnConstrain
78                          (Board),
79                          generateRandomVals
80                          (Board, Size)
81                          ,
82                          generateFields(
83                          Board, Fields
84                          ),
85                          statistics(
86                          runtime, [T1,
87                          _]),
88                          removeBoardFilling
89                          (Board,
90                          EmptyBoard),
91                          printBoard(
92                          EmptyBoard),

```

```

78                                     write( '\n\n'),
79                                     printFieldTable(
80                                         EmptyBoard,
81                                         Fields),
82                                     solveBoard(
83                                         EmptyBoard,
84                                         Fields),
85                                     T is T1 - T0,
86                                     format( 'Creating
                                           _took_~3d_sec
                                           .~n', [T]).

87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102

```

Generations

```

removeBoardFilling([], []).
removeBoardFilling([B | Bs], [EB | EBs]) :-
    removeBoardFillingInRow(B, EB),

removeBoardFillingInRow([], []).
removeBoardFillingInRow([cell(FID, _Val) | Rs], [cell(FID
    , _) | ERs]) :- removeBoardFillingInRow(Rs, ERs).

initBoard(_Size, []).
initBoard(Size, [B | Bs]) :- length(B, Size),

initBoardRow
(

```

		B) ,
103		initBoard (Size ,
		Bs) .
104		
105	initBoardRow ([]) .	
106	initBoardRow ([cell (_FieldID , _Val) Rs]) :- initBoardRow (Rs) .	
107		
108	generateRandomVals (Board , Size) :- getValsList (Board , L) ,	
109		random (1 , Size ,
		X) ,
110		retractall (val (X)) ,
111		random (1 , 3 , Dir)


```
130                                     X1 = 0.
131
132 not(X) :- X, !, fail.
133 not(_X).
134
135 generateFields(Board, Fields) :- generateFields(Board,
        Fields, 1).
136
137 generateFields(Board, [F | Fs], FieldIt) :- not(
        isBoardFilled(Board)),
138
```

139

140

141

142	generateFields(_Board, [], _FieldIt).	
143		
144	isBoardFilled([]).	
145	isBoardFilled([B Bs]) :- isBoardFilledIterRow(B),	isBoardFilled
146		(Bs
147).
148	isBoardFilledIterRow([]).	
149	isBoardFilledIterRow([cell(FID, _) Rs]) :- nonvar(FID),	
150		
151		
152		
153	iterBoard(Board, F, FieldIt) :- field(FieldIt, _Op, _Res)	
154	= F,	repeat
155		,
		random
		(1,
		6,
		OpIt
)
		,
156		length
		(
		Board
		,
		BoardSize
)
		,
157		initField
		(

		OpIt
		,
		F
		,
		BoardSize
		,
		FieldSize
)
		,
158		getFstAvailCe
		(
		Board
		,
		Row
		,
		Col
)
		,
159		getCell
		(
		Board
		,
		Row
		,
		Col
		,
		cell
		(
		FieldIt
		,
		Val
)
)

160		,
		FieldSize1
		is
		FieldSize
		—
		1,
161		prepMakeField
		(
		Board
		,
		Row
		,
		Col
		,
		F
		,
		FieldSize1
		,
		Val
)
		.
162		
163	prepMakeField(Board, Row, Col, F, FieldSize, Val) :—	
164	FieldSize > 0,	

165

166

167

```
168 prepMakeField(_Board, _Row, _Col, field(_, '=', Val),  
    _FieldSize1, Val).
```

169

```
170 initField(1, field(_FieldIt, '+', _Res), BoardSize, Size)  
    :- random(2, BoardSize, Size).
```

```
171 initField(2, field(_FieldIt, '-', _Res), _BoardSize, 2).
```

```
172 initField(3, field(_FieldIt, '*', _Res), BoardSize, Size)  
    :- random(2, BoardSize, Size).
```

```
173 initField(4, field(_FieldIt, '/', _Res), _BoardSize, 2).
```

```
174 initField(5, field(_FieldIt, '=', _Res), _BoardSize, 1).
```

175

```
176 makeField(Board, Row, Col, field(FieldIt, Op, NewAcum),  
177     1, Acum) :- getCell(Board, Row, Col, cell(FieldIt, Val  
    )),
```

```
178  
179 makeField(Board, Row, Col, field(FieldIt, Op, Res),  
    FieldSize, Acum) :- getCell(Board, Row, Col, cell(  
180     FieldIt, Val)),
```

```
181
```

182

183

184

185

```
186
187 getNextCellPos(Board, Row, Col, NewRow, NewCol) :-
    getNextCellPos(Board, Row, Col, NewRow, NewCol, 9).
188 getNextCellPos(_Board, _Row, _Col, _NewRow, _NewCol, 0)
    :- !, fail.
189 getNextCellPos(Board, Row, Col, NewRow, NewCol, _It) :-
    random(1, 4, X),
190
```

```
191 getNextCellPos(Board, Row, Col, NewRow, NewCol, It) :-
    NewIt is It - 1,
192
```

```
193
194 isCellAvailable(Board, Row, Col, X, NewRow, NewCol) :-
195     getTestRow(X, Row, Col, NewRow, NewCol),

196

197
198 getTestRow(1, Row, Col, Row, NewCol) :- NewCol is Col +
199     1.
200 getTestRow(2, Row, Col, NewRow, Col) :- NewRow is Row +
    1.
200 getTestRow(3, Row, Col, Row, NewCol) :- NewCol is Col -
    1,
```



```

201
202
203 getCell([B | _Bs], 1, Col, Cell) :- getCellInRow(B, Col,
    Cell).
204 getCell([_B | Bs], Row, Col, Cell) :- Row1 is Row - 1,
205
206
207 getCellInRow([C | _Cs], 1, C).
208 getCellInRow([_C | Cs], Col, Cell) :- Col1 is Col - 1,
209
210
211 getNewAcum(Acum, '+', Val, NewAcum) :- NewAcum is Acum +
    Val.
212 getNewAcum(Acum, '-', Val, NewAcum) :- min_member(Min, [
    Acum, Val]),

```

213

214

215 `getNewAcum(Acum, '*' , Val , NewAcum) :- NewAcum is Acum *`
 `Val .`

216 `getNewAcum(Acum, '/' , Val , NewAcum) :- min_member(Min, [`
 `Acum, Val]) ,`

217

218

219

X

220

Ne

```
221 getNewAcum(_Acum, '=' , Val, Val).
```

222

```
223 getFstAvailCell(Board, RetRow, RetCol) :- getFstAvailCell  
      (Board, 1, RetRow, RetCol).
```

224

```
225 getFstAvailCell([], _RowIt, _RetRow, _RetCol) :- fail.
```

```
226 getFstAvailCell([B | _Bs], RowIt, RowIt, RetCol) :-  
      getFstAvailCellInRow(B, 1, RetCol).
```

```
227 getFstAvailCell([_B | Bs], RowIt, RetRow, RetCol) :-  
      RowIt1 is RowIt + 1,
```

228

```

229
230 getFstAvailCellInRow ([ , - , - ) :- fail.
231 getFstAvailCellInRow ([ cell(FID, -) | _Rs], ColIt , ColIt)
    :- var(FID) .
232 getFstAvailCellInRow ([_R | Rs], ColIt , RetCol) :- ColIt1
    is ColIt + 1,
233

```

```

234

```

```

235

```

```

236

```

```

237 /*

```

```

    *****

```

```

238 *

```

```

239 * Getters

```

```

240 *

```

```

241 *****

```

```

    */

```

```

242

```

```

243 getFieldCells( , [ ] , [ ] ) .

```

```

244 getFieldCells(FieldID , [B | Bs] , RetList) :-

```

```

    getFieldCellsInRow(FieldID , B, RetList1) ,

```

```

245

```

246

247

248 `getFieldCellsInRow(_FieldID, [], []).`249 `getFieldCellsInRow(FieldID, [cell(FieldID, Val) | Rs],`
RetList) :- `getFieldCellsInRow(FieldID, Rs, RetList1),`

250

251

252 `getFieldCellsInRow(FieldID, [_R | Rs], RetList) :-`
`getFieldCellsInRow(FieldID, Rs, RetList).`

253

254 `getValsList([], []).`255 `getValsList([B | Bs], L) :- getValsListInRow(B, L1),`

256

`getValsList`
`(`
`Bs`
`,`

`L2`
`)`

```

257                                     ,
                                     append
                                     (
                                     L1
                                     ,
                                     L2
                                     ,
                                     L
                                     )
                                     .

258
259 getValsListInRow ([], []).
260 getValsListInRow ([ cell(-, Val) | Rs], L) :-
261     getValsListInRow(Rs, L1),

262
263
264 /*
        *****
265 *
266 * Restrictions
267 *
268 * *****
        */

269
270 imposeDomainConstrain([], -).
271 imposeDomainConstrain([B | Bs], SupLim) :-
        imposeDomainConstrainInRow(B, SupLim),

```

272

273

274 `imposeDomainConstrainInRow([], _).`275 `imposeDomainConstrainInRow([cell(_, Value) | Rs], SupLim)`
`:- Value in 1..SupLim,`

276

277

278 `imposeRowConstrain([]).`279 `imposeRowConstrain([B | Bs]) :- imposeRowConstrain(B, [])`

280

,

`imposeRowCon``(``Bs``)``.`

281

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

283

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

285

```

286
287 imposeColumnConstrain(Board) :- transpose(Board, TBoard),
288

```

```

imposeRowCon
(
  TBoard
)
.

```

```

289
290 imposeFieldConstrain(_Board, []).
291 imposeFieldConstrain(Board, [field(FID, Op, Res) | Fs])
    :- getFieldCells(FID, Board, L),
292

```

```

293

```

```

294
295 applyOpConstrain([], _Op, _Res) :- fail.
296
297 applyOpConstrain(L, '+', Res) :- sum(L, #=, Res).
298
299 applyOpConstrain(L, '-', Res) :- L = [_ , -],
300

```

```

maximum
(
  Max
,

```


		<div>L</div> <div>)</div> <div>,</div>
301		<div>minimum</div> <div>(</div> <div>Min</div> <div>,</div>
		<div>L</div> <div>)</div> <div>,</div>
302		<div>Res</div>
		<div>#=</div>
		<div>Max</div>
		<div>—</div>
		<div>Min</div> <div>.</div>
303		
304	<div>applyOpConstrain(L, ' / ', Res) :- L = [-, -],</div>	
305		<div>maximum</div> <div>(</div> <div>Max</div> <div>,</div>
		<div>L</div> <div>)</div> <div>,</div>
306		<div>minimum</div> <div>(</div> <div>Min</div> <div>,</div>
		<div>L</div> <div>)</div> <div>,</div>

307		Res
		\neq
		Max
		/
		Min
		.
308		
309	<code>applyOpConstrain([L], '=', Res) :- L \neq Res.</code>	
310		
311	<code>applyOpConstrain([L Ls], '*', Res) :- applyOpConstrain(Ls, '*', L, Res).</code>	
312		
313	<code>applyOpConstrain([], '*', Acum, Res) :- Acum \neq Res.</code>	
314		
315	<code>applyOpConstrain([L Ls], '*', Acum, Res) :- Acum1 \neq L * Acum,</code>	
316		
317		
318		
319	<code>/* *****</code>	<code>*****</code>
320	<code>* * Print</code>	
321		

```

322 *
323 *****
    */
324
325 printBoard :- testBoardPrint(Board),
326               printBoard(Board).
327
328 printBoard(Board) :- printTopBorder(Board),
329                       length(Board,
330                               Size),
331                       printBoard(Board
332                               , Size , 1),
333                       printBottomBorder
334                               (Board).
335
336 printTopBorder(Board) :- length(Board, Size), asserta(
337     size(Size)),
338                               write(' _
339                               _ _ '),
340                               write
341                               (' || '
342                               ),
343                               printHorizTopBorder
344                               (Size
345                               ).
346
347 printBoard([], -, -).
348 printBoard([B1, B2 | Bs], Size, RowIt) :- printNumber(
349     RowIt), write(' || '),
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

341		
342		
343		
344		
345	<code>printBoard([B1 Bs], Size, RowIt) :- printNumber(RowIt),</code>	
346	<code> write(' '),</code>	

347

348

349

350

351

352

353

354

355

356

357

```

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
                             ),

```

```

358                                     write(
                                     ' | ',
                                     ),
359                                     printRow
                                     ([
                                     C2
                                     |
                                     Cs
                                     ] ).
360
361 printRow([C1 | Cs]) :- cell(_, Value1) = C1,
362                                     printNumber(
                                     Value1),
363                                     write(' | '),
364                                     printRow(Cs).
365
366 printBottomBorder(Board) :- length(Board, Size), write(' _
_ _ | | '), printHorizBotBorder(Size), write(' _ '),
printBottomNumbers(Size).
367
368 printHorizTopBorder(1) :- write(' === | '), write(' \n ').
369
370 printHorizTopBorder(Size) :- write(' ===== '), Size1 is Size
- 1, printHorizTopBorder(Size1).
371
372 printHorizBotBorder(1) :- write(' === | '), write(' \n ').
373
374 printHorizBotBorder(Size) :- write(' ===== '), Size1 is Size
- 1, printHorizBotBorder(Size1).
375
376 printBottomNumbers(Size) :- write(' _ _ _ '),
printBottomNumbers(Size, 1).
377
378 printBottomNumbers(Size, Size) :- printNumber(Size).
379
380 printBottomNumbers(Size, Number) :- printNumber(Number),
write(' _ '), Number1 is Number + 1, printBottomNumbers(
Size, Number1).
381
382
383 printHorizMidBorder([R1], [R2]) :- cell(FieldID1, _) = R1
,
384

```

```

cell
(
FieldID

```

		,
)
		=
		R2
		,
385		FieldID1
		=
		FieldID
		,
386		write
		(
		,
		--- \
		n
		,
)
		.
387		
388	printHorizMidBorder ([_R1], [_R2]) :- write ('=== \n').	
389		
390	printHorizMidBorder ([R1 R1s], [R2 R2s]) :- cell (
391	FieldID1, _) = R1,	

392

393

394

395

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

397

```
398 printNumber(Number) :- var(Number), write('   ').
```

```
399 printNumber(Number) :- Number > 99, write(Number).
```

```
400 printNumber(Number) :- Number > 9, write('_'), write(
    Number).
```

```
401 printNumber(Number) :- write('_'), write(Number), write(
    '_').
```

402

```
403 printFieldTable(_, []).
```

```
404 printFieldTable(Board, [F | Fs]) :- RowIt = 1,
    printFieldTableAux(Board, F, RowIt), printFieldTable(
    Board, Fs).
```

405

406

```
407 printFieldTableAux([], _, _).
```

```
408 printFieldTableAux([B | _Bs], F, RowIt) :- ColIt = 1,
    findFieldInCell(B, F, ColIt, Col), field(FID, Op, Val)
    = F,
```


409

```
410
411 printFieldTableAux([_B | Bs], F, RowIt) :- RowIt1 is
      RowIt + 1, printFieldTableAux(Bs, F, RowIt1).
412
413 findFieldInCell([], _, _, _) :- fail.
414 findFieldInCell([cell(FID, _) | _Bs], field(FID, _, _),
      ColIt, ColIt).
415 findFieldInCell([_B | Bs], F, ColIt, Col) :- ColIt1 is
      ColIt + 1, findFieldInCell(Bs, F, ColIt1, Col).
416
417 printField(FID, Op, Val, Row, Col) :- write('Field:␣'),
      write(FID), write('␣'),
418
```

wri

419

wri

420

wri

421

422