Easy as ABC: Resolução de Problema de Decisão usando Programação em Lógica com Restrições

Maria Marques & Sofia Reis

Faculdade de Engenharia, Universidade do Porto
14 de Dezembro, 2014
FEUP-PLOG, Turma 3MIEIC03, Grupo Easy as ABC_1
{ei12104,ei12041}@fe.up.pt

Resumo Este artigo serve de apoio ao 2º projeto desenvolvido para a unidade curricular Programação em lógica do 3º ano do Mestrado Integrado em Engenharia Informática e Computação. O projeto consiste na resolução do problema de decisão combinatória *Easy as ABC* usando programação lógica com restrições. O resultado desta implementação é um algoritmo que cria e subsequentemente resolve tabuleiros recorrendo à linguagem CLP utilizando o ambiente SICSTUS. É possível concluir que as restrições são extremamente importantes neste tipo de resoluções, visto que tornam a sua implementação mais rápida e simples. Em puzzles como o Easy As ABC, podemos concluir que quanto maior o tamanho de tabuleiro e o número de restrições maior é o tempo despendido na resolução.

Keywords: Easy as ABC, Prolog, Restrições, CLP, SICSTUS

1 Introdução

O objetivo deste trabalho é gerar e resolver um puzzle lógico *Easy As ABC*, que consiste em inserir todas as letras apenas uma vez em cada linha e coluna da matriz. Este puzzle é gerado dinamicamente, segundo o tamanho das listas recebidas pelo programa e resolvido através da informação contida nessas listas, que são chamadas de dicas do jogo.

Com este projeto, o pretendido é obter os conhecimentos e as boas práticas de programar em prolog com restrições e domínios para resolver problemas. A programação com restrições baseia-se essencialmente em procurar uma solução na qual as restrições utilizadas são satisfeitas.

A nossa motivação na escolha deste tema foi o interesse por puzzles lógicos de natureza não-verbal, como por exemplo o *Sudoku*[1]. Ao longo do curso nunca tivemos oportunidade de solucionar este tipo de problemas e pareceu-nos interessante ter esta primeira experiência fazendo a implementação na linguagem de programação prolog.

Este artigo serve de apoio à solução realizada para o puzzle e menciona os seguintes pontos: primeiro, é feita a descrição em detalhe do puzzle, descrevendo as suas variáveis de decisão e seus domínios, e também a descrição das restrições rígidas e flexíveis do problema. Por fim, neste ponto, é descrita a estratégia de pesquisa utilizada, nomeadamente, ao que diz respeito à ordem das variáveis e valores. Em seguida, é apresentada a visualização da solução mencionando quais os predicados utilizados. E são apresentados os resultados da aplicação em instâncias do problema com diferentes complexidades.

[1] Jogo lógico criado em 1892 que tem como objetivo preencher uma grelha 9x9 com dígitos

2 Descrição do Problema

O puzzle *Easy as ABC*, também conhecido como *Buchstabensalat* ou *End View* é um puzzle lógico. Um puzzle lógico deriva do campo matemático dedução, este tipo de puzzle foi produzido pela primeira vez pelo matemático *Charles Lutwidge Dodgson*, também conhecido como escritor do livro Alice no País das Maravilhas com o pseudónimo de *Lewis Carroll*.

O puzzle consiste num tabuleiro NxN que é preenchido com N-1 letras do alfabeto em cada linha e coluna, ficando assim N espaços em branco. Para o puzzle ser resolvido são apresentadas dicas à volta do tabuleiro, essas dicas indicam a primeira letra que deverá estar no tabuleiro a partir dessa direção. Além das dicas o tabuleiro tem de, para cada coluna e linha, estar preenchido com todas as letras exatamente uma vez.

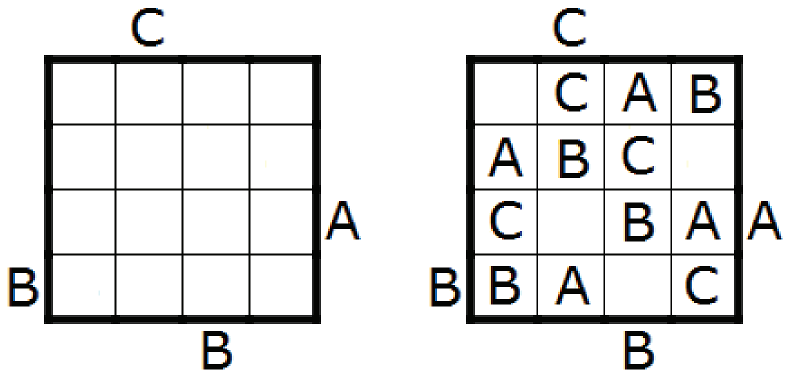

Figura 1: Tabuleiro em situação inicial (com pistas ao redor) e em situação final (problema resolvido)

Na implementação de avaliação da matriz, foi necessário usar o predicado tranpose para calcular a transposta da matriz, e permitir avaliar essa mesma matriz com as restrições através do predicado clueM. Este predicado avalia as dicas da lista que recebe com a primeira coluna da matriz. Sendo por isso necessário calcular a transposta sempre que se quer calcular as restrições através de uma lista de dicas.

3 Abordagem

3.1 Variáveis de Decisão

As variavéis de decisão são as variáveis que calculamos, isto é, às quais fazemos *labeling*. No caso do tabuleiro, cada elemento é uma variável de decisão, visto que usamos uma lista de listas para o representar. Para fazer *labeling* é necessário antes utilizar o *append*.

```
% Solution
solve(L1,L2,L3,L4,Vars):-
...
    append(Vars,L),
    labeling([],L),
...
```

3.2 Restrições

No puzzle *Easy As ABC* todas as restrições são rígidas e não há restrições flexíveis. Na geração dinâmica do tabuleiro, restringimos que todos os elementos de cada linha e coluna têm de ser diferentes:

```
% Generates a dinamic board
gentab([],_).
gentab([H|T],S):-
...
% Puts each row with different elements
    all_different(H),
    gentab(T,S).
```

```
solve(L1,L2,L3,L4,Vars):-
...
    gentab(Vars,S),
% Puts each row with different elements
    transpose(Vars, TVars),
    gentab(TVars,S),
...
```

Além da restrição anterior, é necessário fazer com que a disposição das letras no tabuleiro obdeça às dicas. Para isso, é utilizada a fórmula apresentada em seguida:

$$p_i = 0 \quad \vee \quad x_{i0} = p_i \quad \vee \quad (x_{i0} = 0 \quad \wedge \quad x_{i1} = p_i)$$

Ou o elemento da pista é 0 ou é igual ao primeiro elemento da linha correspondente na matriz do resultado ou o primeiro elemento da linha é zero e o elemento da pista é igual ao segundo elemento da linha correspondente na matriz do resultado.

O predicado responsável por implementar as restrições anteriores é a *clueM* que recebe uma lista de listas que vai ser o tabuleiro final e uma das listas recebidas pelo predicado principal. Sendo que, este predicado é utilizada para todas as listas de dicas. Em seguida, está representado o predicado:

```
clueM([],[]).
clueM([[H1,H2|_]|T],[Ch|Ct]):-
    Ch #= 0 #\/ Ch #= H1 #\/ (H1 #= 0 #/\ Ch #= H2),
    clueM(T,Ct).
```

3.3 Estratégia de Pesquisa

Na estratégia de pesquisa o primeiro ponto a abordar é a utilização do predicado clueM que é responsável por comparar todas as listas de dicas recebidas pelo predicado principal com o tabuleiro. Basicamente o que é verificado é, caso haja pista, se o elemento da pista fica na primeira posição da linha correspondente ou na segunda. Sendo que o tabuleiro é preenchido com N-1 letras em cada linha e consequentemente um espaço fica vazio.

Para o resultado ser obtido de forma mais rápida e com menor utilização de recursos é utilizada a opção ff no labeling.

Labeling ff: (*fail first*): rotular a variável mais à esquerda com menor domínio seguinte, a fim de detectar inviabilidade cedo.

4 Visualização da Solução

Para visualizar o tabuleiro temos um predicado principal chamado *print_end* que recebe as listas de dicas e o tabuleiro final. Este predicado usa outras duas: a *print_clue* responsável por fazer print das dicas contidas nas listas L2 e L4, e a *print_tab* responsável por fazer print do tabuleiro apenas. A *print_tab* usa a *print_line* para imprimir todas as linhas do tabuleiro resolvido e das dicas na vertical, basicamente, as listas L1 e L3.

```
print_end(L1,L2,L3,L4,Vars):-
    nl,
    write(' '),print_clue(L2),nl,
    print_tab(0,Vars,L1,L3),
    write(' '),print_clue(L4).
```

```
print_clue([]).
print_clue([H|L]) :-
    write(' '),
    convert(H,A),write(A),
    print_clue(L).
```

```
print_line([]).
print_line([H|L]) :-
    write('|'),
    convert(H,A),write(A),
    print_line(L).
```

```
print_tab(_,[],[],[]).
print_tab(N,[H|L],[L1|R1],[L3|R3]) :-
    convert(L1,A), write(A),
    print_line(H), write('|'),
    convert(L3,B), write(B),
    nl,
    N2 is N+1,
    print_tab(N2,L,R1,R3).
```

Em seguida apresentamos um exemplo de visualização do tabuleiro:

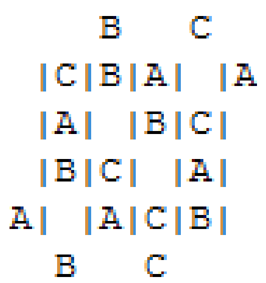

Figura 2: *Output* de tabuleiro com pistas que podem ser visualizadas à volta do tabuleiro

5 Resultados

Realizamos vários testes, com várias complexidades e analisando o resultado de um tabuleiro 4x4, 6x6 e 12x12 chegamos à conclusão de que quanto maior é o tabuleiro maior é o numero de restrições criadas (Constraints created) mas também o número de opções inviáveis (Prunings) é maior. Tal como o número de restrições criadas é maior, também o número de resumptions é. Em termos, de tempo todas as soluções escolhidas obteram tempo menor de 0.0s. Concluindo, quanto maior for o tamanho do tabuleiro, maior será o número de restrições percorridas. Estas conclusões podem ser retiradas através da observação das imagens seguintes.

```
Time: 0.0s

Resumptions: 544
Entailments: 162
Prunings: 296
Backtracks: 3
Constraints created: 96

    B   C
 |C|B|A| |A
 |A| |B|C|
 |B|C| |A|
A| |A|C|B|
  B   C
V = [[3,2,1,0],[1,0,2,3],[2,3,0,1],[0,1,3,2]] ?
```

Figura 3: Resultado: tabuleiro 4x4

```
Time: 0.0s

Resumptions: 558
Entailments: 144
Prunings: 387
Backtracks: 0
Constraints created: 144

    B   C
 | |B|D|C|E|A|A
 |C| |A|D|B|E|
 |D|A|E|B| |C|
A|A|E|B| |C|D|
 |E|C| |A|D|B|
 |B|D|C|E|A| |
  B   C
V = [[0,2,4,3,5,1],[3,0,1,4,2,5],[4,1,5,2,0,3],[1,5,2,0,3,4],[5,3,0,1,4,2],[2,4,3,5,1|...]] ?
```

Figura 4: Resultado: tabuleiro 6x6

```
Time: 0.0s

Resumptions: 2335
Entailments: 519
Prunings: 2394
Backtracks: 9
Constraints created: 288

    B D C
 |C|B|D| |E|F|G|H|I|J|K|A|A
 |J| |A|C|B|D|E|F|G|H|I|K|
 |D|C|I|J| |K|A|B|E|F|G|H|
A|A|H| |F|C|B|J|K|D|G|E|I|
 |I|J|H|G|A| |F|C|K|E|B|D|D
 |E|I|B|H|F|C|D| |A|K|J|G|
 |F|D|E|B|G|H|K|A|J|I|C| |
 |G|E|F|K|H|J| |I|C|D|A|B|B
A| |A|K|E|I|G|C|J|H|B|D|F|
 |K|F|G|D|J|A|I|E|B| |H|C|
 |H|G|J|A|K|I|B|D| |C|F|E|
 |B|K|C|I|D|E|H|G|F|A| |J|
  B   C   D
V = [[3,2,4,0,5,6,7,8,9|...],[10,0,1,3,2,4,5,6,7|...],[4,3,9,10,0,11,1,2|...],
[1,8,0,6,3,2,10|...],[9,10,8,7,1,0|...],[5,9,2,8,6|...],[6,4,5,2|...],[7,5,6|...],[0,1|...],[...|...]|...] ?
```

Figura 5: Resultado: tabuleiro 12x12

6 Conclusões e Trabalho Futuro

Com este projeto foi possível perceber a utilidade de restrições em Prolog, o impacto que elas têm na simplificação da resolução de problemas que noutras linguagens seriam muito mais complexos. O objetivo proposto, de permitir lidar com tamanhos diferentes de tabuleiros e números diferentes de peças, foi atingido. A solução que propusemos tem a limitação de não fazer a geração dinâmica de problemas, mas por outro lado é feita a visualização da solução em modo de texto e de forma precetível com um output de um tabuleiro.

A maior dificuldade foi perceber como utilizar a matriz para verificar as restrições, o que foi ultrapassado com a utilização do predicado transpose responsável por calcular a matriz tranposta.

Concluindo, conseguimos criar uma solução fiável de resolução de puzzles *Easy As ABC*.

Referências

1. First April Sprint Test — Logic Masters India Puzzle - Puzzle Booklet, April (2014)
2. Vinckensteiner — Puzzles and Brain Games, `http://raetsel.vinckensteiner.com/logik/en/abc/`
3. SWI Prolog — Predicate labeling/2, `http://www.swi-prolog.org/pldoc/man?predicate=labeling/2`
4. Wikipedia — Logic Puzzle, `http://en.wikipedia.org/wiki/Logic_puzzle`
5. Wikipedia — Buchstabensalat, `http://en.wikipedia.org/wiki/Buchstabensalat_%28logic_puzzle%29`
6. Wikipedia — Sudoku, `http://en.wikipedia.org/wiki/Sudoku`

Anexo: Código

```
% ***************************************************************************
% *                              EASY_AS_ABC                                *
% ***************************************************************************
% * Realizado por:                                                          *
% *           Maria Marques  - ei12104@fe.up.pt                             *
% *           Sofia Reis - ei12041@fe.up.pt                        MIEIC14 *
% ***************************************************************************
:-use_module(library(clpfd)).
:-use_module(library(lists)).

% Generates a dinamic board
gentab([],_).
gentab([H|T],S):-
        length(H, S),
        S1 #= S - 1,
        domain(H,0,S1),
        all_different(H),
        gentab(T,S).

% Constraints
clueM([],[]).
clueM([[H1,H2|_]|T],[Ch|Ct]):-
        Ch #= 0 #\/ Ch #= H1 #\/ (H1 #= 0 #/\ Ch #= H2),
        clueM(T,Ct).

solve(L1,L2,L3,L4,Vars):-
        conv(L1,NL1),
        reverse(NL1,RNL1),
        conv(L2,NL2),
        reverse(NL2,RNL2),
        conv(L3,NL3),
        reverse(NL3,RNL3),
        conv(L4,NL4),
        reverse(NL4,RNL4),
        solution(RNL1,RNL2,RNL3,RNL4,Vars).

% Solution
solution(NL1,NL2,NL3,NL4,Vars):-
        length(NL1, S),
        length(Vars, S),
        gentab(Vars,S),
        transpose(Vars, TVars),
        gentab(TVars,S),
        clueM(Vars,NL1),
        clueM(TVars,NL2),
        reverse(TVars,RTVars),
        transpose(RTVars,TRTVars),
```

```
        clueM(TRTVars,NL3),reverse(Vars, RVars),
        transpose(RVars, TRVars),
        clueM(TRVars,NL4),
        append(Vars,L),
        reset_timer,
        labeling([ff],L),
        print_time,
        fd_statistics,
        print_end(NL1,NL2,NL3,NL4,Vars).

reset_timer :- statistics(walltime,_).
print_time :-
        statistics(walltime,[_,T]),
        TS is ((T//10)*10)/1000,
        nl, write('Time: '), write(TS), write('s'), nl, nl.

% Print final board
print_end(L1,L2,L3,L4,Vars):-
        nl,
        write(' '),print_clue(L2),nl,
        print_tab(0,Vars,L1,L3),
        write(' '),print_clue(L4).

print_clue([]).
print_clue([H|L]) :-
        write(' '),
        convert(H,A),write(A),
        print_clue(L).

print_line([]).
print_line([H|L]) :-
        write('|'),
        convert(H,A),write(A),
        print_line(L).

print_tab(_,[],[],[]).
print_tab(N,[H|L],[L1|R1],[L3|R3]) :-
        convert(L1,A), write(A),
        print_line(H), write('|'),
        convert(L3,B), write(B),
        nl,
        N2 is N+1,
        print_tab(N2,L,R1,R3).

conv([],_).
conv([L|T],NL):-
        convert(A,L),
        append(NewNL,[A],NL),
        conv(T,NewNL).
```

```
convert(0,' ').
convert(1,'A').
convert(2,'B').
convert(3,'C').
convert(4,'D').
convert(5,'E').
convert(6,'F').
convert(7,'G').
convert(8,'H').
convert(9,'I').
convert(10,'J').
convert(11,'K').
convert(12,'L').
convert(13,'M').
convert(14,'N').
convert(15,'O').
convert(16,'P').
convert(17,'Q').
convert(18,'R').
convert(19,'S').
convert(20,'T').
convert(21,'U').
convert(22,'V').
convert(23,'W').
convert(24,'X').
convert(25,'Y').
convert(26,'Z').
```