Programação em Lógica

Início	Quarta, 8 de Janeiro de 2020 às 14:08
Estado	Prova submetida
Data de submissão:	Quarta, 8 de Janeiro de 2020 às 16:06
Tempo gasto	1 hora 58 minutos
Nota	**19,70** de um máximo de 20,00 (**99**%)

Informação

Destacar pergunta

Considere o seguinte programa em Prolog:

```
:- use_module(library(lists)).
:- use_module(library(clpfd)).

programa(L):-
    length(L, 10),
    domain(L, 1, 20),
    labeling([], L),
    sumlist(L, 110),
    check(L).

check([]).
check([H|T]):-
    diff(H, T),
    check(T).

diff(_, []).
diff(E, [H|T]):-
    E \= H,
    diff(E, T).
```

Pergunta 1

Respondida Pontuou 1,000 de 1,000 Destacar pergunta

Indique o que faz este programa e comente quanto à sua eficiência.

O programa gera uma lista com 10 elementos, todos com valores entre 1 e 20, tal que todos os números da sequência sejam diferentes e tal que a sua soma seja igual a 110.
A eficiência desta implementação não é a melhor. Uma vez que as restrições do problema são impostas depois de se chamar o predicado labeling (que atribui um valor válido a cada variável), é utilizado um processo de Generate&Test, em vez de Constrain&Generate (que seria pôr as restrições antes do labeling). A quantidade de retrocessos e testes poderá ser substancialmente maior com Generate&Test.

Pergunta 2

Respondida Pontuou 1,000 de 1,000 Destacar pergunta

Qual a dimensão do espaço de procura deste problema?

Selecione uma opção de resposta:

- a. 10^{20}
- b. $10 * 20$
- c. 20^{10}
- d. $10 + 20$
- e. [Não Responder]

Pergunta 3

Respondida Pontuou 2,500 de 2,500 Destacar pergunta

Proponha uma solução alternativa para resolver o problema que tire partido do mecanismo '*constrain and generate*' disponível no paradigma de PLR, implementando-a no predicado ***melhor/1***.

```
:- use_module(library(clpfd)).
:- use_module(library(lists)).

melhor(L) :-
  length(L, 10),
  domain(L, 1, 20),
  all_distinct(L),
  sum(L, #=, 110),

  labeling([], L).
```

Necessário passar as verificações para restrições antes da chamada ao labeling:

```
melhor(L):-
  length(L, 10),
  domain(L, 1, 20),
  sum(L, #=, 110),
  all_distinct(L),
  labeling([], L).
```

Pergunta 4

Respondida Pontuou 1,500 de 1,500 Destacar pergunta

Altere o programa que produziu na alínea anterior de forma a minimizar a gama de valores da lista (ie, minimizar a diferença entre os valores mais alto e mais baixo). Implemente estas alterações no predicado ***equilibrado/1***.

```
:- use_module(library(clpfd)).
:- use_module(library(lists)).

equilibrado(L) :-
  length(L, 10),
  domain(L, 1, 20),
  all_distinct(L),
  sum(L, #=, 110),

  minimum(Min, L),
```

Necessário encontrar os valores máximo e mínimo, minimizando a diferença entre os dois:

```
equilibrado(L):-
  length(L, 10),
  domain(L, 1, 20),
  sum(L, #=, 110),
  all_distinct(L),
  minimum(Min, L),
```

```
    maximum(Max, L),
    Diff #= Max - Min,
    labeling([minimize(Diff)], L).
```

Pergunta 5

Respondida Pontuou 3,000 de 3,000 Destacar pergunta

Implemente o predicado ***tresQuadrados/3***, que determine 3 números distintos (entre 1 e 200), tais que a soma dos três números resulte num quadrado perfeito, e o produto de quaisquer dois deles, adicionado ao terceiro, resulte também em quadrados perfeitos.

Nota: evite simetrias nas soluções obtidas.

Exemplo:

```
| ?- tresQuadrados(A, B, C).
A = 4,
B = 12,
C = 33 ?
```

(4+12+33=49 (7^2) ; 4*12+33=81 (9^2) ; 4*33+12=144 (12^2) ; 12*33+4=400 (20^2))

```
:- use_module(library(clpfd)).
:- use_module(library(lists)).

tresQuadrados(A, B, C) :-
    domain([A, B, C], 1, 200),

    Aux1 #>= 0, Aux2 #>= 0 , Aux3 #>= 0, Aux4 #>= 0,

    all_distinct([A, B, C]),
```

Verificação de quadrados perfeitos:

```
    A + B + C #= W * W,
    A * B + C #= X * X,
    A * C + B #= Y * Y,
    B * C + A #= Z * Z,
```

De forma a evitar simetrias, basta garantir ordenação dos valores:

```
    A #< B, B #< C
```

(esta restrição garante já também que os valores são distintos entre si)

Pergunta 6

Respondida Pontuou 1,500 de 1,500 Destacar pergunta

Considere o seguinte programa em Prolog:

```
problem([A, B, C]):-
    domain([A, B, C], 1, 5),
    A #> B,
    B #> C,
    % Aqui
    A + B + C #= A * B * C,
    labeling([], [A, B, C]).
```

Qual o estado do (hiper-)grafo de restrições associado a este problema após declaração de domínios e colocação das duas primeiras restrições (local assinalado no código)?

(sugestão: use o PowerPoint, ou o Paint, para desenhar o grafo com as variáveis, domínios associados e restrições, e faça upload da imagem do grafo)

Considerando que após a colocação de cada restrição dá-se a propagação dessa restrição nos domínios das variáveis, de modo a manter a consistência, o grafo será: (ver imagem)

Untitled.png

Inicialmente as três variáveis não estão ligadas entre si e têm todas domínio 1..5

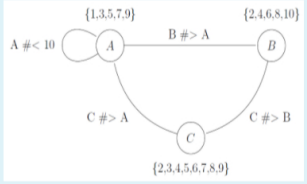
Ao colocar a restrição A #> B é criada uma ligação entre A e B, e os domínios destas variáveis são alterados de acordo com esta restrição (A in 2..5, B in 1..4).

Ao colocar a restrição B #> C é criada uma ligação entre B e C, e os domínios das variáveis são alterados: B in 2..4, C in 1..3. Dado que o domínio de B foi alterado, há propagação pela primeira restrição, e o domínio de A será atualizado para 3..5.

Pergunta 7

Respondida Pontuou 1,500 de 1,500 Destacar pergunta

Considere o seguinte grafo de restrições representativo de um problema de satisfação de restrições:



Este grafo é:

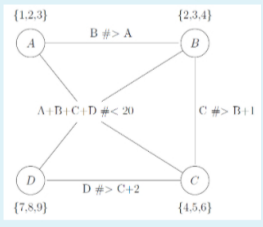
Selecione uma opção de resposta:

- a. 1-consistente, 2-consistente e 3-consistente.
- b. [Não Responder]
- c. 1-consistente e 3-consistente, mas não 2-consistente.
- d. 1-consistente, mas não 2-consistente ou 3-consistente.
- e. 1-consistente e 2-consistente, mas não 3-consistente.

Pergunta 8

Respondida Pontuou 1,400 de 1,500 Destacar pergunta

Considere o seguinte hiper-grafo de restrições representativo de um problema de satisfação de restrições:



Assumindo que este hiper-grafo representa o estado de um problema antes da chamada ao *labeling*, e que a chamada é feita com os seguintes parâmetros:

```
labeling([ffc, step, down], [A,B,C,D])
```

qual o estado do grafo após o primeiro passo de seleção de variável e valor, e respectiva propagação de restrições?

(sugestão: use o PowerPoint, ou o Paint, para desenhar o grafo com as variáveis, domínios associados e restrições, e faça upload da imagem do grafo)

ffc faz com que a variável escolhida seja a variável com menor domínio, desempatando com a escolha da que tem mais restrições suspensas;

step é a escolha por omissão, sendo que faz com que o valor atribuído a essa variável seja a escolha binária entre X #= B e X #\= B, ondeB é a lower ou upper bound de X;
down faz com que o domínio de valores dessa variável seja explorado por ordem descendente.

A variável escolhida será a B, uma vez que ela e a C têm o menor domínio e mais restrições suspensas, mas B está mais a esquerda na chamada ao labeling;

Untitled.png

A opção *ffc* seleciona a variável com o domínio mais restringido, desempatando com a variável envolvida em mais restrições, e depois pela mais à esquerda. Neste caso, as variáveis têm todas domínios com o mesmo tamanho (3 valores); B e C estão ambas envolvidas em mais restrições (3 restrições); e B é a variável mais à esquerda na chamada ao *labeling*, logo a escolhida. A opção *step*, combinada com *down*, significa que será escolhido o valor mais elevado do domínio de B (4).

Ao escolher este valor, haverá propagação; pela restrição entre B e C, C toma o valor 6; com nova propagação pela restrição entre C e D, D toma o valor 9. Com nova propagação (pela restrição de soma), conclui-se que A teria de ser no máximo 0, ou seja, B=4 não é uma solução viável. Este valor será removido do domínio de B, e por propagação, o valor 3 é também removido do domínio de A.

Informação

Destacar pergunta

O professor Ronaldo dos Sete necessita de preparar um teste, composto por 7 questões, uma por cada um dos 7 temas da sua cadeira. Ele dispõe de uma base de dados com várias questões por cada tema. Cada questão tem um identificador único, um identificador do tema da questão, um nível de dificuldade (entre 1 e 3) e um tempo esperado de realização (em minutos).

```
%question(IDQuestão, IDTema, NivelDificuldade, TempoEstimado).
question( 1, 1, 1, 3).        question( 2, 1, 2, 5).        question( 3, 1, 3, 8).
question( 4, 2, 1, 4).        question( 5, 2, 2, 6).        question( 6, 2, 3, 9).
question( 7, 3, 1, 4).        question( 8, 3, 2, 6).        question( 9, 3, 3, 9).
question(10, 4, 1, 6).        question(11, 4, 2, 9).        question(12, 4, 3, 12).
question(13, 5, 1, 6).        question(14, 5, 2, 9).        question(15, 5, 3, 12).
question(16, 6, 1, 6).        question(17, 6, 2, 9).        question(18, 6, 3, 12).
question(19, 7, 1, 6).        question(20, 7, 2, 9).        question(21, 7, 3, 12).
```

É desejado que o tempo total de realização do teste seja de 55 minutos, e que sejam escolhidas pelo menos duas questões de cada nível de dificuldade.

O Ronaldo pede-lhe a sua ajuda para implementar o predicado ***makeTest(+NCategories, +TotalTime, +MinQuestionsPerLevel, -Questions)*** que recebe como parâmetros o número de categorias (questões) desejadas, a duração total desejada para a prova e o número mínimo de questões por nível de dificuldade, retornando os identificadores das questões selecionadas.

Exemplo:

```
| ?- makeTest(7, 55, 2, Q).
Q = [1,4,8,11,14,18,21] ?
```

Pergunta 9

Respondida Pontuou 1,000 de 1,000 Destacar pergunta

Descreva como vai modelar o problema, as variáveis de decisão a usar, os seus domínios e respetivo significado.

As variáveis de decisão que utilizarei serão uma lista de questões que serão escolhidas, ordenadas por tema, sendo que a pergunta escolhida para o tema i estará na posição i da lista (a começar em 1). As variáveis de decisão irão ter domínio igual a um intervalo de 1 ao número total de perguntas (obviamente que irão ser implementadas restrições de modo a que na posição i só seja possível escolher perguntas do tema i). Em termos de restrições, a lista de variáveis de domínio iria ser iterada, de modo a escolher para cada tema uma pergunta, indo somando o tempos que demora a fazer numa variável. No fim, essa variável teria de ser igual a 55 (ou TotalTime). Depois, com o uso de restrições materializadas para verificar e somar o numero de perguntas de um determinado grau de dificuldade, iria-se implementar restrições de modo a que esse número, para cada grau, seja maior ou igual a MinQuestionsPerLevel.

Pergunta 10

Respondida Pontuou 3,800 de 4,000 Destacar pergunta

Implemente o predicado ***makeTest/4*** como descrito acima.

```
makeTest(NCategories, TotalTime, MinQuestionsPerLevel, Questions) :-
  length(Questions, NCategories),
  findall(Theme, question(_, Theme, _, _), Themes),
  findall(Dif, question(_, _, Dif, _), Difs),
  findall(Time, question(_, _, _, Time), Times),

  % irrelevante, mas ajuda a ser mais eficiente acho eu
  all_distinct(Questions),

  get_questions(Questions, 1, Themes, Times, SumTimes),

  SumTimes #= TotalTime,

  constrain_dif_types(Questions, Difs, MinQuestionsPerLevel),
```

É necessário manter informação relativa aos IDs, tempos e níveis de dificuldade das questões selecionadas. De forma a evitar *backtracking* (e soluções *G&T*), os factos *question* não devem ser consultados na fase de colocação de restrições, mas apenas no início do processamento.

Exemplo de possível solução:

```
makeTest(NCategories, TotalTime, MinQuestionsPerLevel, Questions):-
  length(Questions, NCategories),      % uma questão por cada categoria
  domainsTimesDiffs(Questions, 1, Times, Diffs),      %obter listas de variáveis que representam ID, tempo e dificuldade

  % Restrição de níveis de dificuldade
  global_cardinality(Diffs, [1-A, 2-B, 3-C]),
  A #>= MinQuestionsPerLevel,
   B #>= MinQuestionsPerLevel,
   C #>= MinQuestionsPerLevel,

  % Restrição de tempo total
  sum(Times, #=, TotalTime),

  append([Questions, Times, Diffs], Vars),
  labeling([], Vars).

% Coloca os domínios das variáveis, garantindo já que as questões são todas diferentes, e é selecionada apenas uma por categoria.
domainsTimesDiffs([], _, [], []).
domainsTimesDiffs([Q|Qs], Num, [Time|Times], [Diff|Diffs]):-
  findall(ID-T-D, question(ID, Num, D, T), ITDs),
  msplit(ITDs, IDs, Ts, Ds),      %Separar ID, tempo e dificuldade, de forma a poder aplicar o predicado element

  % garantir que ID, tempo e dificuldade se referem à mesma questão
  element(Idx, IDs, Q),      % domínio do ID fica restringido aos IDs das questões da categoria existentes na base de dados
  element(Idx, Ts, Time),
  element(Idx, Ds, Diff),
  Num1 is Num + 1,
  domainsTimesDiffs(Qs, Num1, Times, Diffs).

msplit([], [], [], []).
msplit([ID-T-D|R], [ID|IDs], [T|Ts], [D|Ds]):-
  msplit(R, IDs, Ts, Ds).
```

Pergunta 11

Respondida Pontuou 0,500 de 0,500 Destacar pergunta

Descreva as alterações que faria no seu programa de forma a que caso não haja solução com o tempo exato pretendido, seja devolvida a solução com o tempo mais próximo.

No programa é gerada uma variável de domínio que representa a soma dos tempos de todas as perguntas, sendo que no fim, essa variável é restringida, sendo que tem de ser igual ao valor passado através de TotalTime.
Em vez disso, o que se poderá fazer é ter uma variável que representa a diferença absoluta entre o tempo pretendido TotalTime e a soma dos tempos das perguntas, sendo que o objetivo será minimizar essa variável, de modo a que a diferença entre os dois tempos seja a mais pequena possível (no melhor dos casos a diferença é 0 e os dois tempos são iguais).

Pergunta 12

Respondida Pontuou 1,000 de 1,000 Destacar pergunta

Implemente as alterações descritas na alínea anterior no predicado ***makeTest(+NCategories, +TotalTime, +MinQuestionsPerLevel, -Questions, -TestTime)***, que retorna adicionalmente em *TestTime* a duração do teste gerado.

```
makeTest(NCategories, TotalTime, MinQuestionsPerLevel, Questions, TestTime) :-
```

```
makeTest(NCategories, TotalTime, MinQuestionsPerLevel, Questions, TestTime) :-
  length(Questions, NCategories),
  findall(Theme, question(_, Theme, _, _), Themes),
  findall(Dif, question(_, _, Dif, _), Difs),
  findall(Time, question(_, _, _, Time), Times),

  % irrelevante, mas ajuda a ser mais eficiente acho eu
  all_distinct(Questions),

  get_questions(Questions, 1, Themes, Times, TestTime),

  TimeDifference #= abs(TestTime - TotalTime),

  constrain_dif_types(Questions, Difs, MinQuestionsPerLevel),
```

É apenas necessário minimizar a diferença absoluta entre o tempo do teste gerado e o tempo desejado.

```
makeTest(NCategories, TotalTime, MinQuestionsPerLevel, Questions, TestTime):-
  ...
  % Restrição de tempo do teste
  sum(Times, #=, TestTime),
  Diff #= abs(TotalTime - TestTime),

  % minimizar a diferença
  labeling([[minimize(Diff)], Vars).
```

Terminar revisão

◄ Mini-Teste 1 (2019-11-20)

Ir para...

SICStus Prolog ►