Departamento de Engenharia Informática

Primeiro Trabalho

Autores: A49877 Diogo Miguel Rebelo de Sousa
A51548 João Alexandre Rebelo de Sousa

Grupo 12

Relatório para a Unidade Curricular de Logica Computacional

Professor(a): Nuno Leite

24–10–2025

<< Esta página foi intencionalmente deixada em branco

Índice

Índice

Exercicios

- ## 1º exercicio:

Esta fase corresponde ao primeiro exercício sobre Predicate Logic, contém 2 alíneas:

Alínea a):

Este alínea é sobre um programa de Prolog composto pelas seguintes cláusulas:

C1:vL conc([], L, L).

C2:vX,vL2 conc([X], L2, [X | L2]).

C3:vX,vL1,vL2,vL3 conc([X | L1], L2, [X | L3]) :-

conc(L1, L2, L3).

Objetivo deste exercício é provar que a cláusula C2 é redundante. Para que uma cláusula seja redundante é necessário provar que remove-la do programa não altera o valor logico, ou seja, podemos obtê-la a partir das outras cláusulas.

Para isso é necessário realizar os seguintes passos:

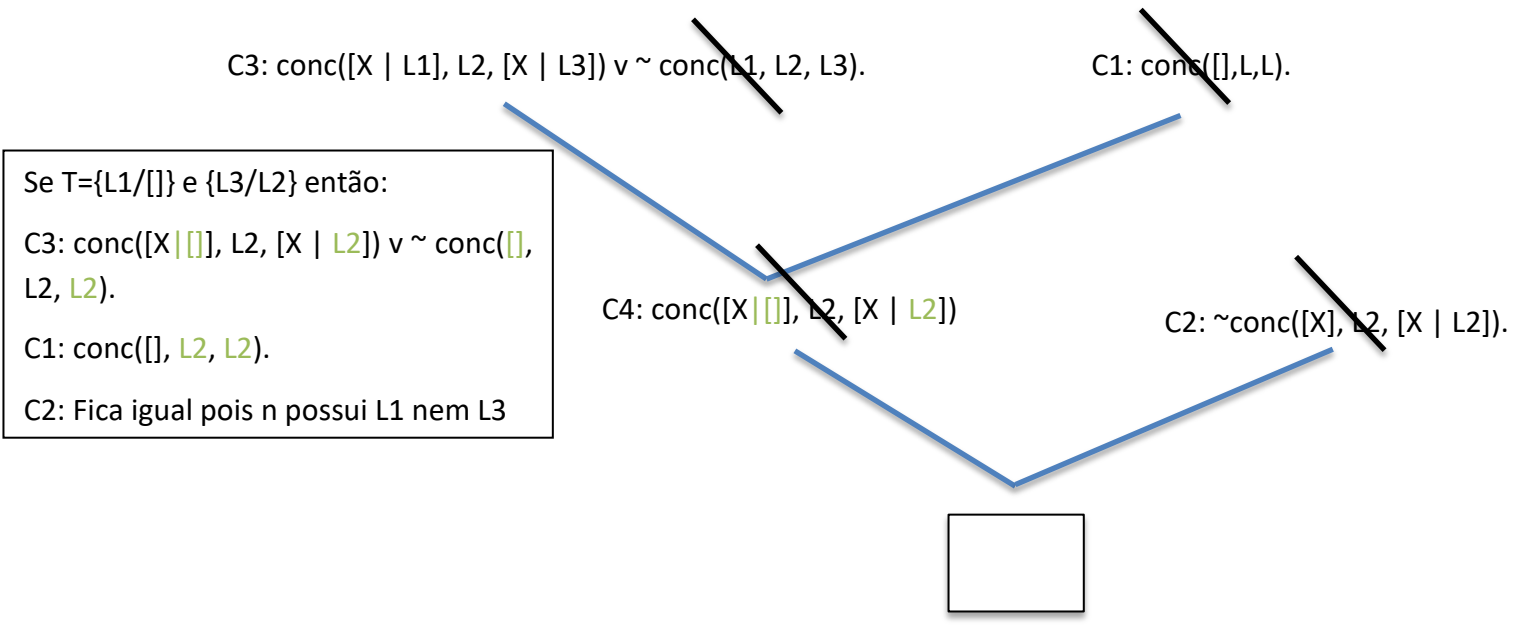
➔ 1º Transformar na forma CNF:

C1 e C2 já estão na forma CNF, C3 aplicando a transformação passa a ser conc([X | L1], L2, [X | L3]) v ~ conc(L1, L2, L3).

➔ 2º Mostrar que C2 é derivável a partir de C1 e C3:

C1 v C3 v ~C2 = □

Árvore de dedução (Usando o principio da resolução):



Se T={L1/[]} e {L3/L2} então:

C3: conc([X|[]], L2, [X | L2]) v ~ conc([], L2, L2).

C1: conc([], L2, L2).

C2: Fica igual pois n possui L1 nem L3

R: Como se conseguiu cortar tudo então C2 é redundante, podemos obter a clausula C2 a partir de C1 e C3.

Alínea b):

Esta alínea é igual a anterior com a diferença que em vez de ser utilizado cálculo de predicados é usado dedução em Prolog.

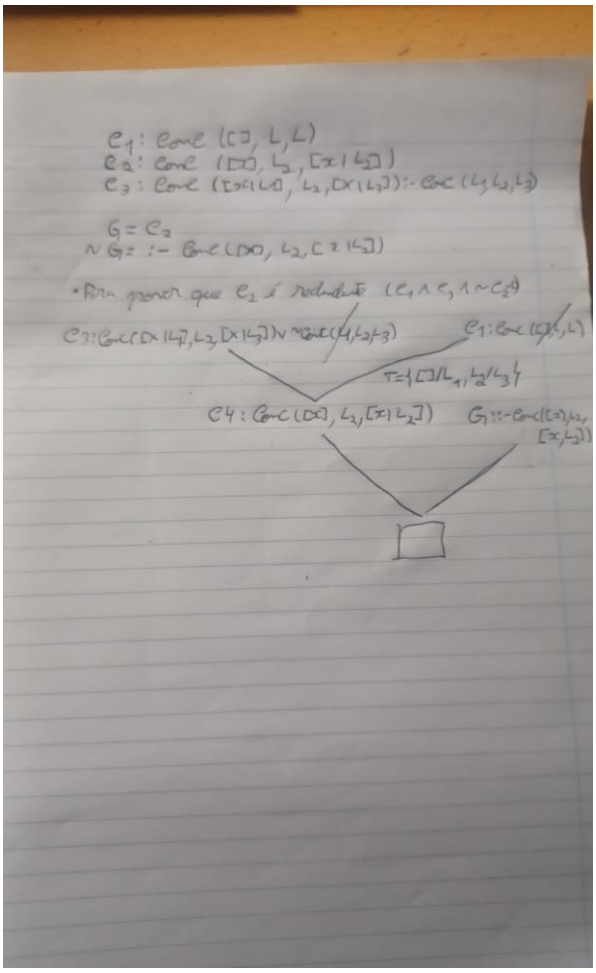


Fig-1: Resolução do exercício 1 alínea b

- **2ºExercicio:**

Esta fase corresponde ao segundo exercício, contem 4 alíneas. Para este exercício é necessário estar ciente das seguintes ideias:

→O modelo de execução do Prolog utiliza o algoritmo DFS(depth first search). Explora cada ramo da árvore de execução até ao fim antes de considerar alternativas, avançando sempre pelo predicado mais à esquerda e recorrendo ao backtracking quando ocorre uma falha.

→O cut (!) remove ramos da árvore.

Alínea a):

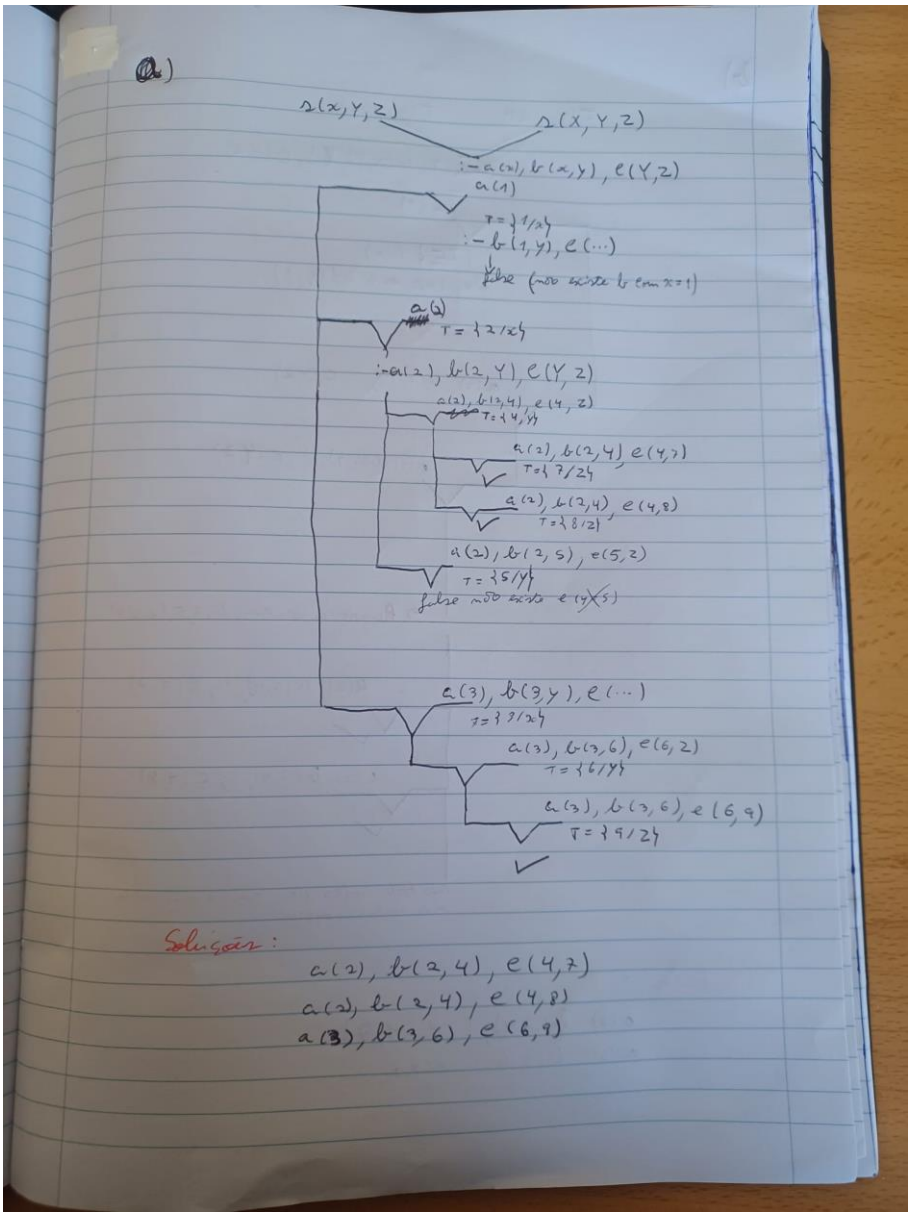


Fig -2: Resolução do exercício 2 alínea a).

- A figura 2 mostra o percurso tomado pelo Prolog para a clausula s(X, Y, Z) :- a(X), b(X, Y), c(Y, Z). .
    - Primeiro o Prolog começa por a(x) das quais x pode ter os seguintes valores (1,2,3).
    - Começa por fazer uma unificação, processo fundamental do Prolog para fazer corresponder duas expressões (por exemplo: a(X) = a(1)), ao fazer isso obtemos a clausula :-a(1),b(1,y),… , mas b(1,…) não existe logo falha e o Prolog faz backtracking (o processo de desfazer as unificações feitas até ao ultimo ponto, voltar para atrás para o ponto onde existiam alternativas e tentar a próxima alternativa) neste caso a escolha de a(x). Assim, o Prolog tenta a alternativa seguinte: a(X) = a(2), gerando a substituição X = 2. Agora tenta b(2, Y) e encontra duas possibilidades: Y = 4 e Y = 5. Para cada possibilidade, tenta depois c(Y, Z). Para Y = 4, encontra duas soluções (Z = 7 e Z = 8); para Y = 5, não encontra nenhuma.
      Depois volta novamente ao backtracking e tenta a(3), levando à solução final (X,Y,Z) = (3,6,9).

Alínea b):

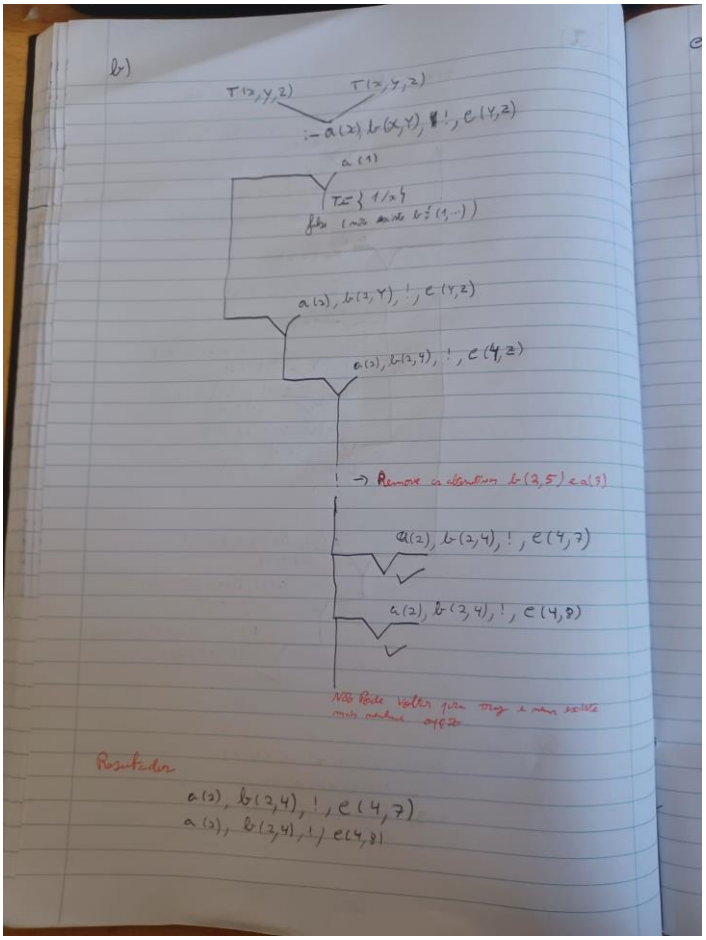


Fig3:Resoluçao do exercício 2 da alinea b)

- A figura 3 mostra o percurso tomado pelo Prolog para a clausula `t(X, Y, Z) :- a(X), b(X, Y),!, c(Y, Z).` .

Neste caso, o comportamento é semelhante ao da alínea anterior até ao momento em que o predicado b(X,Y) encontra uma solução válida. O Prolog começa por tentar a(1), mas tal como antes falha em b(1,Y). Faz então backtracking e tenta a(2). Para X = 2, obtém b(2,4) como a primeira solução do predicado. É neste momento que entra o operador de corte !. Quando o Prolog alcança o corte, todas as alternativas anteriores são descartadas, incluindo:

- outras alternativas de b(2,Y) (por exemplo b(2,5)),
- e outras alternativas de a(X) (por exemplo a(3)).

A partir daqui o Prolog fica "comprometido" com a escolha (X,Y) = (2,4) e só tenta alternativas para o predicado c(Y,Z), que produz Z = 7 e Z = 8.

Assim, devido ao cut, o Prolog não procura quaisquer outras combinações de X e Y, e as únicas soluções possíveis são (2,4,7) e (2,4,8).

Alínea c):

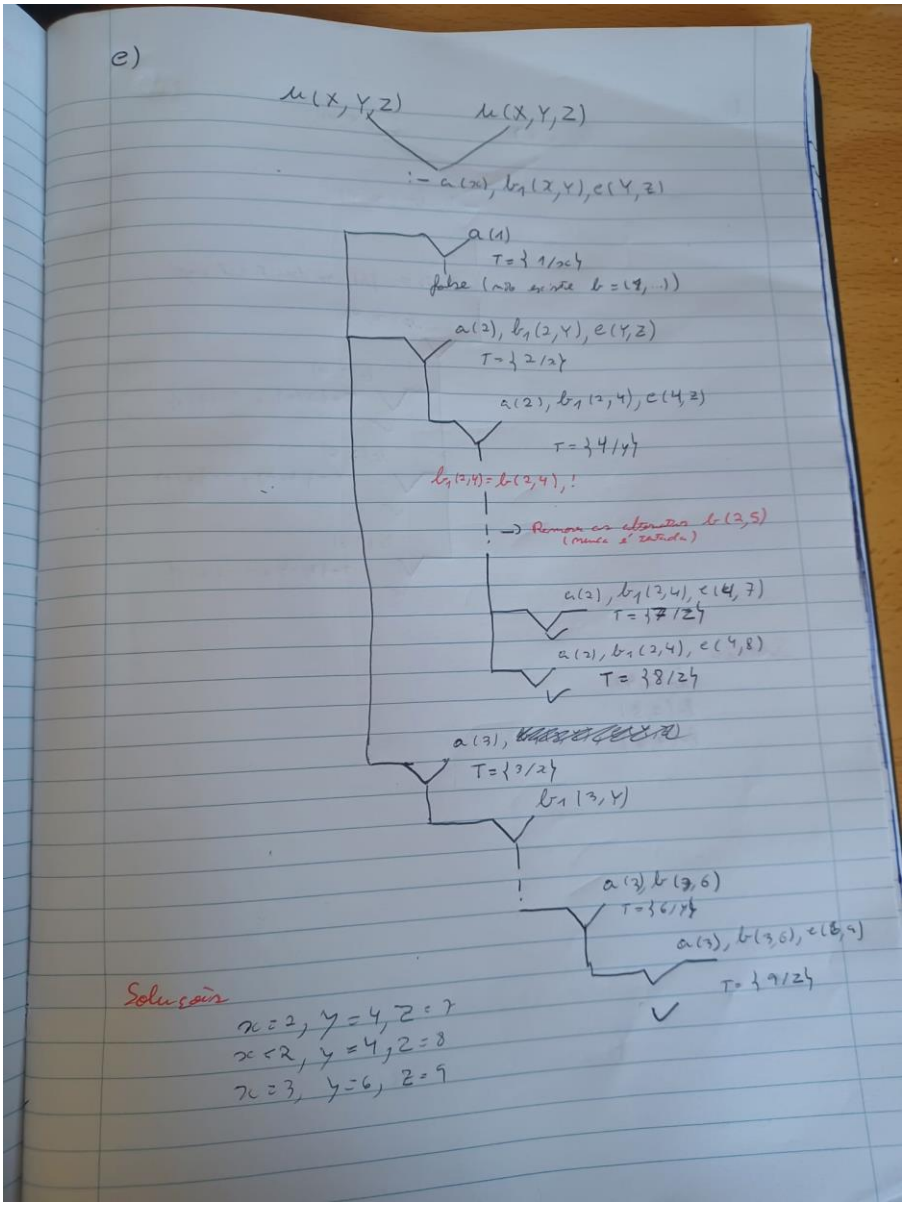


Fig-4: Resolução do exercício 2 alínea c)

- A figura 3 mostra o percurso tomado pelo Prolog para a clausula u(X, Y, Z) :- a(X), b1(X, Y), c(Y, Z)..

Aqui, o corte não está na cláusula principal, mas sim dentro do predicado auxiliar b1/2. O Prolog começa novamente por tentar a(1), que falha, e depois tenta a(2). Para X = 2, b1(2,Y) tenta unificar com b(2,Y). A primeira solução possível é Y = 4, e ao encontrá-la o predicado b1/2 executa o corte (!). Este corte elimina todas as alternativas de b(2,Y), impedindo o Prolog de tentar b(2,5), mas não elimina alternativas de a(X), porque o corte está dentro de b1, e não no predicado u/3.

Assim, para X=2, Y=4, o Prolog tenta c(4,Z) e encontra duas soluções (Z=7 e Z=8). Depois, devido ao backtracking ao nível de a(X), o Prolog tenta também a(3) e obtém b1(3,6) (única solução, seguida de corte), e depois c(6,9).

No total, mesmo com o corte interno, o Prolog produz as mesmas três soluções que em s/3, embora seguindo um caminho mais restrito dentro do predicado b1/2.

Alínea d):

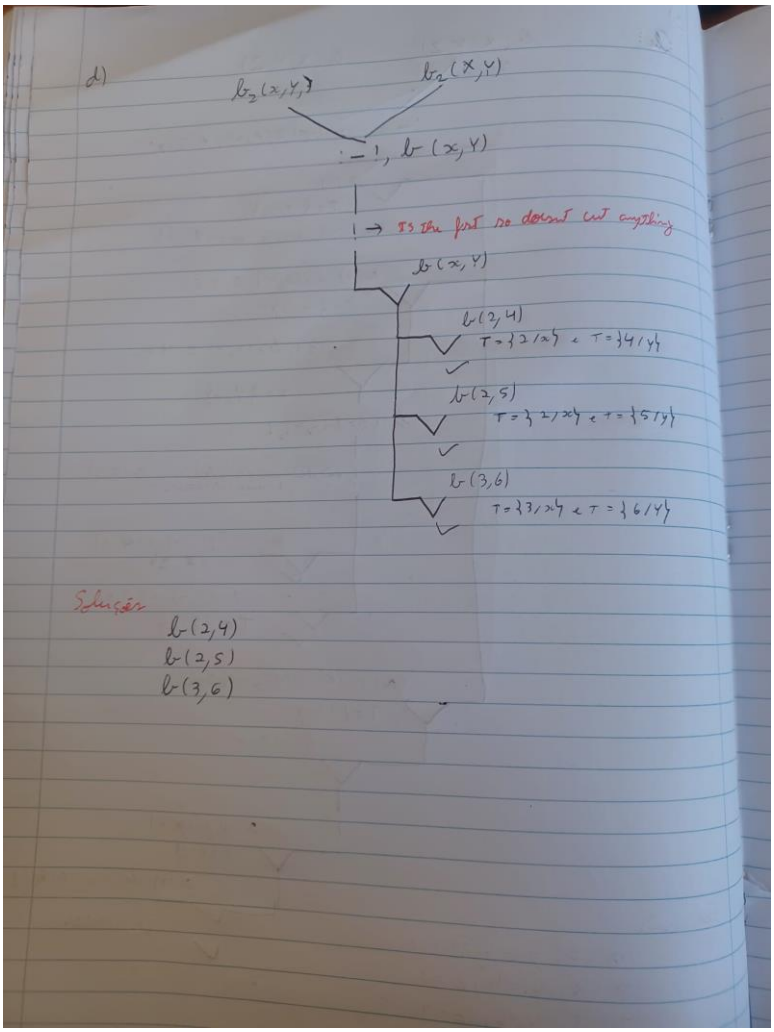


Fig-4: Resolução do exercício 2 alínea d)

- A figura 4 mostra o percurso tomado pelo Prolog para a clausula b2(X, Y) :- !, b(X, Y)..

O corte aparece como primeiro objetivo da cláusula. Quando o Prolog entra em b2/2, o corte é executado imediatamente. Este corte serve apenas para impedir o Prolog de tentar outras cláusulas alternativas de b2/2, caso existam. Depois do corte, o Prolog tenta b(X,Y) e obtém todas as soluções disponíveis: (2,4), (2,5) e (3,6).

Note-se que o corte não afeta as alternativas dentro de b/2, porque estas surgem depois do cut e não antes dele.

Assim, o Prolog devolve exatamente todas as soluções de b/2.

- ## 3ºExercico:

  O objetivo deste exercício é o desenvolvimento do jogo connect4 em Prolog.

  Foram utilizados os principais predicados:

  →start. : Inicia o jogo chamando o predicado game_init

  →empty_board(Board).: cria um boad e retorna-o na variável Board.

  →print_board(Board).: recebe um board e escreve-o na consola

  →init_turn(P1,P2).: pede aos utilizadores para escolherem uma peça e retorna um P que contem o player associado com a cor da peça escolhida pelo mesmo.

  →start_game(Board, P1, P2, Current).: recebe um Board, dois players com as cores associadas e uma variável Current que possui o jogador que vai jogar agora. Este predicado chama outro predicado chamado insert que contem a logica para um player inserir uma peça. No final este predicado faz um "if" que verifica se algum dos users ganhou ou se houve um empate, caso tenha acontecido alguma destas duas, é realizado um cut que diz ao Prolog para ignorar as outras alternativas fazendo com que o jogo acabe pois n existem mais nenhuma alternativa a partir daí.

  - Para compilar fazemos a instrução ?- [main].
  - Para rodar fazemos a instrução ?-main.

Conclusão:

Este trabalho permitiu um aprofundamento nos conhecimentos de Prolog e tambem mais pratica para a realização de futuros testes sobre esta matéria.

Bibliografia:

Slides disponibilizados pelo professor.