Exercícios de Lógica para Programação

Ana Cardoso-Cachopo

Maio de 2014

Conteúdo

Prefácio

Este documento contém uma compilação de exercícios para a disciplina de Lógica para Programação da LEIC e é baseado no livro "Lógica e Raciocínio" do Professor João Pavão Martins.

A maior parte dos exercícios foi criada por mim especificamente para as aulas práticas ou para as provas de avaliação da disciplina e outros foram tirados de livros ou artigos acerca da matéria em questão.

Existem dois exercícios cujo enunciado foi feito pelo Professor João Pavão Martins nas aulas sobre os sistemas de dedução natural da lógica proposicional e da lógica de primeira ordem. Esses exercícios estão assinalados com a etiqueta (JPM).

O Professor João Cachopo, para além de discutir comigo algumas das respostas dos exercícios, ajudou-me a fazer em LaTeX as figuras dos vários capítulos, tornando esta compilação muito mais apresentável.

Obviamente, a responsabilidade por quaisquer erros ou gralhas que esta compilação de exercícios possa ter é inteiramente minha.

Correcções de gralhas ou sugestões de melhorias podem ser enviadas para o meu endereço de email: `acardoso@tecnico.ulisboa.pt`.

Proposta de exercícios para cada aula de problemas

Secção	Exercícios mais relevantes
Argumentos e Validade	1.1 e 1.2
LP — Sistema de Dedução Natural	2.1, 2.2.9, 2.2.11, 2.3.1, 2.3.4
LP — Tabelas de Verdade	3.1 a 3.6
LP — Resolução	4.1, 4.2.1, 4.2.3, 4.5, 4.4.1, 4.3.1, 4.3.11
LP — BDDs	5.1 a 5.4
LP — OBDDs	6.1 e 6.2
LP — SAT	7.1 a 7.6
LPO — Sistema de Dedução Natural	8.1 a 8.3
LPO — Sistema Semântico	9.2 a 9.4
LPO — Representação	10.1 e 10.2
LPO — Resolução	11.1 a 11.4, 11.8
Prog. em Lógica — Resolução SLD; Árvores SLD	12.1 a 12.4
Prolog — Árvores de Refutação; Listas	13.1 a 13.3
Prolog — Operadores Pré-definidos	14.1 a 14.6
Prolog — Corte; Negação	15.1 a 15.16

1 Argumentos e Validade

Sumário:

- Apresentação
- Trazer para as aulas os enunciados e as resoluções dos exercícios
- Argumentos e validade

Resumo:

Um *argumento* é representado por um par (Δ, α), em que Δ é um conjunto de proposições (premissas) e α é uma proposição (conclusão).

Uma *proposição* é uma frase declarativa que pode ser *verdadeira* ou *falsa*.

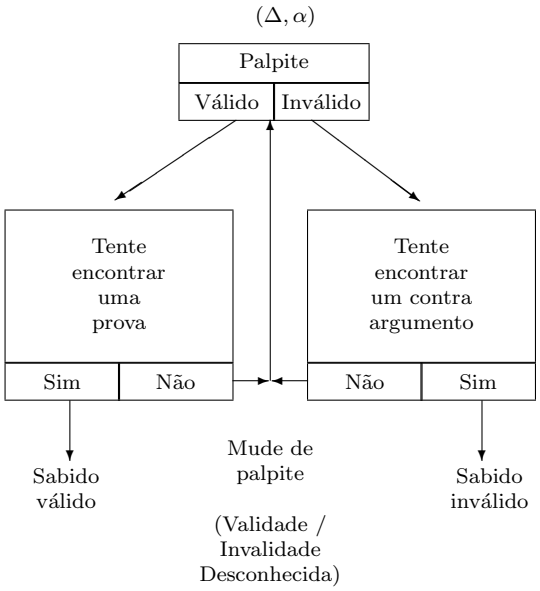
Um argumento é *válido* (podendo também dizer-se que as premissas implicam semanticamente a conclusão, ou que a conclusão é uma consequência semântica das premissas) quando é logicamente impossível ter todas as premissas verdadeiras e a conclusão falsa, ou seja, quando todos os modelos das premissas também são modelos da conclusão. É *inválido* caso contrário.

Princípio da irrelevância do valor lógico: excepto no caso em que as premissas são todas verdadeiras e a conclusão é falsa, a verdade ou falsidade das proposições que constituem um argumento não é relevante para determinar a validade ou invalidade do argumento.

Princípio da forma: se dois argumentos têm a mesma forma, então estes são ambos válidos ou ambos inválidos.

Um *contra-argumento* é um argumento com a mesma forma do argumento inicial, cujas premissas sejam todas verdadeiras e a conclusão seja falsa.

A figura seguinte ilustra uma metodologia para provar a validade/invalidade de um argumento e foi retirada do livro do Professor Pavão Martins.



Os componentes de uma lógica são:

- A *linguagem*, que é definida através de um conjunto de regras que especificam as *fbfs*.
- O *sistema dedutivo*, que contém um conjunto de regras para a manipulação dos símbolos existentes na linguagem, as regras de inferência. Estas regras não fazem parte da linguagem da lógica mas falam sobre as entidades existentes na linguagem, ou seja, pertencem à meta-linguagem da lógica.
- O *sistema semântico*, que especifica as condições sob as quais as proposições (ou fbfs), são verdadeiras ou são falsas. A semântica é baseada no conceito de interpretação, que permite determinar os valores lógicos das proposições.

Quando $(\Delta \vdash \alpha)$, o argumento (Δ, α) é *demonstrável* usando o sistema sintáctico (em inglês "provable").

Quando $(\Delta \models \alpha)$, o argumento (Δ, α) é *válido* usando o sistema semântico.

Uma lógica é *correcta* se qualquer argumento demonstrável usando o seu sistema dedutivo é válido de acordo com a sua semântica.

Uma lógica é *completa* se qualquer argumento válido de acordo com a sua semântica é demonstrável usando o seu sistema dedutivo.

Exercício 1.1
Usando apenas a informação que está explícita, diga, justificando, se os seguintes argumentos são válidos ou são inválidos:

1. Peregrino Cinzento é Gandalf
   Mithrandir é Gandalf
   ∴ Peregrino Cinzento é Mithrandir
2. Mithrandir é um feiticeiro
   Mithrandir é Gandalf
   ∴ Gandalf é um feiticeiro
3. Os orcs são feios
   ∴ Os orcs são feios
4. Nemo é um peixe
   Dori é um peixe
   ∴ Nemo é Dori
5. Os tubarões são carnívoros
   Os tubarões não são vegetarianos
   O Bruce é vegetariano
   ∴ O Bruce não é tubarão
6. Os peixes são animais
   ∴ Os tubarões são animais

Resposta:

Pretendemos saber se cada argumento satisfaz a definição de argumento válido: vamos ver se é possível ter a conclusão falsa e as premissas verdadeiras. Se for possível, o argumento não satisfaz a definição e por isso é inválido; são não for possível, o argumento satisfaz a definição e por isso é válido.

Podemos resolver este exercício usando diagramas de Venn ou usando a *forma* dos argumentos e partindo do princípio que a noção de igualdade (com as suas propriedades) está definida na lógica que estivermos a usar.

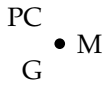
Também podemos fazer as provas "directamente", partindo do princípio que todas as premissas são verdadeiras e testando se a conclusão pode ser falsa; ou por redução ao absurdo, partindo do princípio que a conclusão é falsa e vendo se é possível todas as premissas serem verdadeiras sem dar origem a uma contradição.

Para além destas maneiras de provar se um argumento é válido ou não, também o podemos fazer partindo de um palpite, como está no livro. Se o palpite for que o argumento é válido, tentamos provar que é impossível ter todas as premissas verdadeiras e a conclusão falsa. Se conseguirmos, o argumento é sabido válido. Se não conseguirmos, mudamos o palpite para inválido e tentamos encontrar um contra-argumento, isto é, argumento com a mesma forma mas que nós saibamos que não é válido. Se conseguirmos, o argumento é sabido inválido. Se não conseguirmos provar que o argumento é válido nem inválido, o argumento tem validade desconhecida.

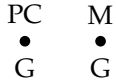
Vamos usar várias alternativas para resolver cada um dos exercícios. Normalmente, deve ser usada apenas uma delas.

1. Peregrino Cinzento é Gandalf
   Mithrandir é Gandalf
   ∴ Peregrino Cinzento é Mithrandir

(a) Diagrama de Venn, prova directa:
Para Peregrino Cinzento ser Gandalf, ambos têm que ser representados pelo mesmo ponto. Para Mithrandir ser Gandalf, ambos têm que ser representados pelo mesmo ponto. Logo, é impossível que Peregrino Cinzento não seja Mithrandir. Assim, o argumento é válido.



(b) Diagrama de Venn, prova por redução ao absurdo:
Para Peregrino Cinzento não ser Mithrandir, têm que ser representados por pontos diferentes. Para Peregrino Cinzento ser Gandalf, ambos têm que ser representados pelo mesmo ponto. Para Mithrandir ser Gandalf, ambos têm que ser representados pelo mesmo ponto. Como Gandalf não pode ser representado simultaneamente por dois pontos diferentes, é impossível ter simultaneamente a conclusão falsa e todas as premissas verdadeiras, o que significa que o argumento é válido.



(c) Forma do argumento, prova directa:
Para Peregrino Cinzento ser Gandalf, têm que ser iguais $PC = G$. Para Mithrandir ser Gandalf, têm que ser iguais $G = M$. Logo, $PC = G = M$ e pela transitividade da igualdade, é impossível que Peregrino Cinzento não seja igual a Mithrandir. Assim, o argumento é válido.

(d) Forma do argumento, prova por redução ao absurdo:
Para Peregrino Cinzento não ser Mithrandir, têm que ser diferentes $PC \neq M$. Para Peregrino Cinzento ser Gandalf, têm que ser iguais $PC = G$. Para Mithrandir ser Gandalf, têm que ser iguais $M = G$. Temos $PC \neq M, PC = G, M = G$. Como Gandalf não pode ser diferente de si mesmo, é impossível ter simultaneamente a conclusão falsa e todas as premissas verdadeiras, o que significa que o argumento é válido.

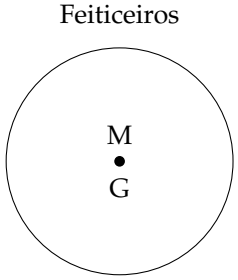
(e) Pelo algoritmo do livro:
Palpite: o argumento é válido.
Resolução: tentar encontrar uma prova. A prova pode ser qualquer uma das apresentadas anteriormente.
Conclusão: como conseguimos encontrar uma prova, o argumento é sabido válido.

Nota: todas as justificações anteriores para a validade deste argumento poderiam ser usadas da mesma maneira para o argumento seguinte, pois não dependem das entidades mencionadas.

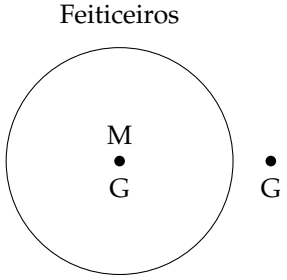
Peregrino Cinzento é Pato Donald
Pato Donald é Super Homem
$\therefore$ Peregrino Cinzento é Super Homem

2. Mithrandir é um feiticeiro
Mithrandir é Gandalf
$\therefore$ Gandalf é um feiticeiro

(a) Diagrama de Venn, prova directa:
Para Mithrandir ser um feiticeiro, o conjunto dos feiticeiros tem que o conter. Para Mithrandir ser Gandalf, ambos têm que ser representados pelo mesmo ponto. Logo, Gandalf também está obrigatoriamente contido no conjunto dos feiticeiros. Assim, o argumento é válido.



(b) Diagrama de Venn, prova por redução ao absurdo:
Para Gandalf não ser feiticeiro, não pode estar contido no conjunto dos feiticeiros. Para Mithrandir ser um feiticeiro, o conjunto dos feiticeiros tem que o conter. Para Mithrandir ser Gandalf, ambos têm que ser representados pelo mesmo ponto. Como Gandalf não pode ser representado simultaneamente por dois pontos diferentes, é impossível ter simultaneamente a conclusão falsa e todas as premissas verdadeiras, o que significa que o argumento é válido.



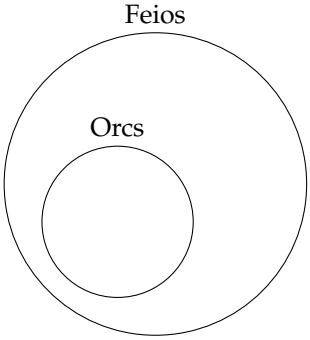
(c) Forma do argumento, prova directa:
Para Mithrandir ser um feiticeiro, $Feiticeiro(M)$. Para Mithrandir ser Gandalf $M = G$. Logo, é possível substituir M por G na primeira fórmula e temos $Feiticeiro(G)$, o que faz com que seja impossível ter a conclusão falsa, ou seja, $\neg Feiticeiro(G)$. Assim, o argumento é válido.

(d) Forma do argumento, prova por redução ao absurdo:
Para Gandalf não ser feiticeiro, temos $\neg Feiticeiro(G)$. Para Mithrandir ser um feiticeiro, temos $Feiticeiro(M)$. Para Mithrandir ser Gandalf $M = G$ e por isso, $Feiticeiro(G)$. Como isto é contraditório com primeira fórmula, é impossível ter simultaneamente a conclusão falsa e todas as premissas verdadeiras, o que significa que o argumento é válido.

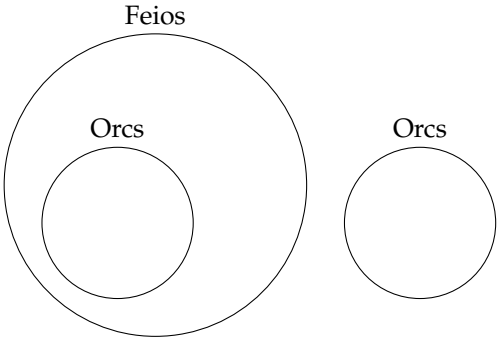
(e) Pelo algoritmo do livro:
Palpite: o argumento é válido.
Resolução: tentar encontrar uma prova. A prova pode ser qualquer uma das apresentadas anteriormente.
Conclusão: como conseguimos encontrar uma prova, o argumento é sabido válido.

3. Os orcs são feios
$\therefore$ Os orcs são feios

(a) Diagrama de Venn, prova directa:
Para os orcs serem feios, o conjunto que os representa tem que estar contido no conjunto que representa os feios. Para a conclusão ser falsa, é necessário que os orcs não sejam feios, isto é, que não estejam contidos no conjunto dos feios. Como isto é impossível dada a primeira frase, o argumento é válido.



(b) Diagrama de Venn, prova por redução ao absurdo:
Vamos supôr que os orcs não são feios, isto é, que o conjunto que os representa não está contido no conjunto que representa os feios. Posto isto, é possível ter a premissa verdadeira, isto é, que os orcs sejam feios, ou seja, que estejam contidos no conjunto dos feios? Como é impossível que o conjunto dos orcs esteja contido e não esteja contido no conjunto dos feios, o argumento é válido.



(c) Forma do argumento, prova directa:
Para os orcs serem feios, temos $\forall(x)[Orc(x) \rightarrow Feio(x)]$. Logo, é impossível termos a conclusão falsa, ou seja $\neg\forall(x)[Orc(x) \rightarrow Feio(x)]$. Assim, o argumento é válido.

(d) Forma do argumento, prova por redução ao absurdo:
Vamos supôr que a conclusão é falsa, isto é, que $\neg\forall(x)[Orc(x) \rightarrow Feio(x)]$. Neste caso, é impossível que a premissa $\forall(x)[Orc(x) \rightarrow Feio(x)]$ seja verdadeira. Logo, como é impossível ter simultaneamente todas as premissas verdadeiras e a conclusão falsa, o argumento é válido.

(e) Pelo algoritmo do livro:
Palpite: o argumento é válido.
Resolução: tentar encontrar uma prova. A prova pode ser qualquer uma das apresentadas anteriormente.
Conclusão: como conseguimos encontrar uma prova, o argumento é sabido válido.

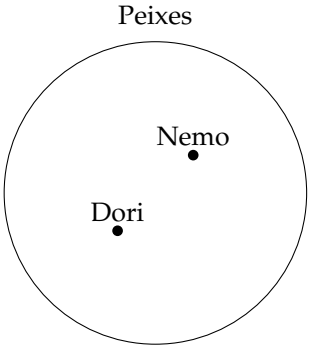
Nota: todas as justificações anteriores para a validade deste argumento poderiam ser usadas da mesma maneira para o argumento seguinte, pois não dependem do significado das palavras usadas nem da opinião do leitor acerca dos orcs.

Os orcs são lindos
$\therefore$ Os orcs são lindos

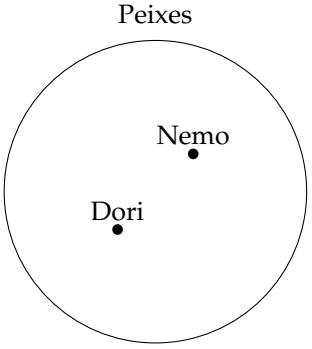
4. Nemo é um peixe
Dori é um peixe
$\therefore$ Nemo é Dori

(a) Diagrama de Venn, prova directa:
Para Nemo ser um peixe, tem que estar contido no conjunto dos peixes. Para Dori ser um peixe, tem que estar contida no conjunto dos peixes. Pode ser igual ou diferente do Nemo. Logo, é possível ter todas as premissas verdadeiras e a conclusão falsa, isto

é, Nemo não ser Dori. Assim, o argumento não é válido.



(b) Diagrama de Venn, prova por redução ao absurdo:
Vamos supôr que a conclusão é falsa, isto é, que Nemo não é Dori. É possível que ambos sejam peixes? Sim, não há nada que impeça o conjunto dos peixes de conter os dois. Logo, como é possível ter simultaneamente todas as premissas verdadeiras e a conclusão falsa, o argumento não é válido.



(c) Forma do argumento, prova directa:
Para Nemo ser peixe, temos $Peixe(Nemo)$. Para Dori ser peixe, temos $Peixe(Dori)$. Nada nos impede de ter $Nemo \neq Dori$. Logo, como é possível ter simultaneamente todas as premissas verdadeiras e a conclusão falsa, o argumento não é válido.

(d) Forma do argumento, prova por redução ao absurdo:
Vamos supôr que a conclusão é falsa, isto é, que $Nemo \neq Dori$. É possível que ambos sejam peixes? Sim, não há nada que impeça de ter simultaneamente $Peixe(Nemo)$ e $Peixe(Dori)$. Logo, como é possível ter simultaneamente todas as premissas verdadeiras e a conclusão falsa, o argumento não é válido.

(e) Pelo algoritmo do livro:
Palpite: o argumento não é válido.
Resolução: tentar encontrar um contra-argumento, isto é, um argumento com a mesma forma mas que nós saibamos que é inválido.

2 é um número
5 é um número
$\therefore$ 2 é 5

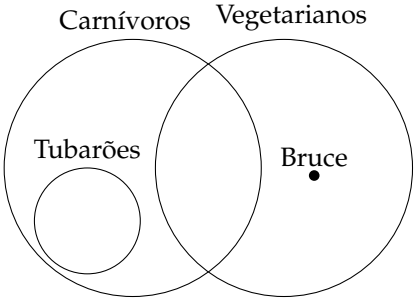
Nós sabemos que 2 não é 5, logo, este argumento tem todas as premissas verdadeiras e a conclusão falsa, por isso não é válido.
Conclusão: como conseguimos encontrar um contra-argumento, o argumento é sabido inválido.

Nota: O argumento seguinte também não é válido, apesar de ter as mesmas premissas mas a conclusão ser a negação da conclusão do anterior. Neste caso, para provar a sua invalidade deveríamos escolher representar o $Nemo$ e a $Dori$ no mesmo ponto. O que isto significa é que por sabermos que ambos são peixes não podemos ter a certeza que são o mesmo nem que são diferentes.

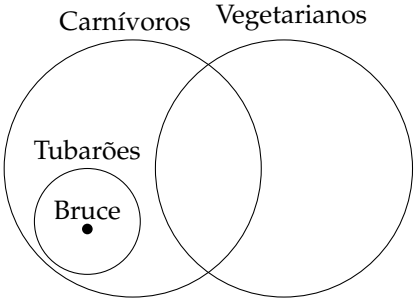
Nemo é um peixe
Dori é um peixe
$\therefore$ Nemo não é Dori

5. Os tubarões são carnívoros
   Os tubarões não são vegetarianos
   O Bruce é vegetariano
   $\therefore$ O Bruce não é tubarão

   (a) Diagrama de Venn, prova directa:
   Para os tubarões serem carnívoros, o conjunto que representa os tubarões tem que estar contido no conjunto que representa os carnívoros. Para os tubarões não serem vegetarianos, o conjunto que representa os tubarões não pode estar contido no conjunto que representa os vegetarianos, embora os carnívoros e os vegetarianos se possam intersectar, sem problemas. Para o Bruce ser vegetariano, tem que estar contido no conjunto que representa os vegetarianos. Com isto, é impossível que o Bruce esteja contido no conjunto que representa os tubarões. Logo, o argumento e válido.



   (b) Diagrama de Venn, prova por redução ao absurdo:
   Vamos supôr que o Bruce é um tubarão, o que significa que tem que estar contido no conjunto que representa os tubarões. Para os tubarões serem carnívoros, o conjunto que representa os tubarões tem que estar contido no conjunto que representa os carnívoros. Para os tubarões não serem vegetarianos, o conjunto que representa os tubarões não pode estar contido no conjunto que representa os vegetarianos. Para o Bruce ser vegetariano, tem que estar contido no conjunto que representa os vegetarianos. Isto é impossível, uma vez que ele é um tubarão e os tubarões não são vegetarianos. Logo, como é impossível ter simultaneamente todas as premissas verdadeiras e a conclusão falsa, o argumento é válido.



   (c) Forma do argumento, prova directa:
   Para os tubarões serem carnívoros, temos $\forall(x)[Tubarao(x) \rightarrow Carnivoro(x)]$. Para os tubarões não serem vegetarianos, temos $\forall(x)[Tubarao(x) \rightarrow \neg Vegetariano(x)]$. Para o Bruce ser vegetariano, temos $Vegetariano(Bruce)$. Dadas estas premissas, é obrigatório que a conclusão seja verdadeira, pois se ela fosse falsa e tivéssemos $Tubarao(Bruce)$ ele não seria vegetariano e teríamos um a contradição. Logo, o argumento e válido.

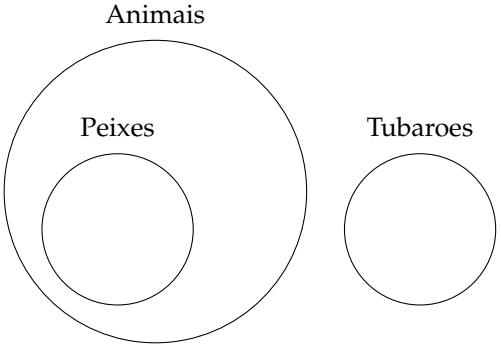
   (d) Forma do argumento, prova por redução ao absurdo:
   Vamos supôr que a conclusão é falsa, ou seja, que o Bruce é um tubarão $Tubarao(Bruce)$. Para os tubarões serem carnívoros, temos $\forall(x)[Tubarao(x) \rightarrow Carnivoro(x)]$. Para os tubarões não serem vegetarianos, temos $\forall(x)[Tubarao(x) \rightarrow \neg Vegetariano(x)]$. Para o Bruce ser vegetariano, temos $Vegetariano(Bruce)$. Isto é impossível, uma vez que ele é um tubarão e os tubarões não são vegetarianos e por isso também temos

$\neg Vegetariano(Bruce)$ que dá uma contradição. Logo, como é impossível ter simultaneamente todas as premissas verdadeiras e a conclusão falsa, o argumento é válido.

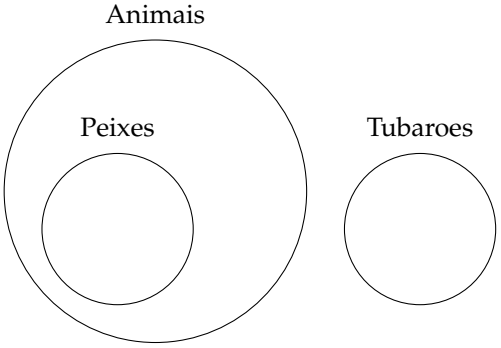
(e) Pelo algoritmo do livro:
Palpite: o argumento é válido.
Resolução: tentar encontrar uma prova. A prova pode ser qualquer uma das apresentadas anteriormente.
Conclusão: como conseguimos encontrar uma prova, o argumento é sabido válido.

6. Os peixes são animais
$\therefore$ Os tubarões são animais

(a) Diagrama de Venn, prova directa:
Para os peixes serem animais, têm que estar contidos no conjunto dos animais. Com base nesta premissa, nada nos impede de ter o conjunto dos tubarões fora do conjunto dos animais. Logo, é possível ter todas as premissas verdadeiras e a conclusão falsa, isto é, os tubarões não serem animais. Assim, o argumento não é válido.



(b) Diagrama de Venn, prova por redução ao absurdo:
Vamos supôr que a conclusão é falsa, isto é, que os tubarões não são animais. É possível que os peixes sejam animais? Sim, não há nada que impeça o conjunto dos peixes de estar contido no conjunto dos animais. Logo, como é possível ter simultaneamente todas as premissas verdadeiras e a conclusão falsa, o argumento não é válido.



(c) Forma do argumento, prova directa:
Para os peixes serem animais, temos $\forall(x)[Peixe(x) \rightarrow Animal(x)]$. Nada nos impede de ter $\neg\forall(x)[Tubarao(x) \rightarrow Animal(x)]$. Logo, como é possível ter simultaneamente todas as premissas verdadeiras e a conclusão falsa, o argumento não é válido.

(d) Forma do argumento, prova por redução ao absurdo:
Vamos supôr que a conclusão é falsa, isto é, que $\neg\forall(x)[Tubarao(x) \rightarrow Animal(x)]$. É possível que os peixes sejam animais, isto é, $\forall(x)[Peixe(x) \rightarrow Animal(x)]$? Sim, não há nada que nos impeça de ter simultaneamente estas duas fórmulas. Logo, como é possível ter simultaneamente todas as premissas verdadeiras e a conclusão falsa, o argumento não é válido.

(e) Pelo algoritmo do livro:
Palpite: o argumento não é válido.
Resolução: tentar encontrar um contra-argumento, isto é, um argumento com a mesma

forma mas que nós saibamos que é inválido.

Os peixes são animais
∴ Os números são animais

Nós sabemos que os números não são animais, logo, este argumento tem todas as premissas verdadeiras e a conclusão falsa, por isso não é válido.
Conclusão: como conseguimos encontrar um contra-argumento, o argumento é sabido inválido.

Exercício 1.2
Sempre que for possível, dê exemplos de argumentos válidos e inválidos com:

- As premissas verdadeiras e a conclusão verdadeira;
- As premissas verdadeiras e a conclusão falsa;
- As premissas falsas e a conclusão verdadeira;
- As premissas falsas e a conclusão falsa.

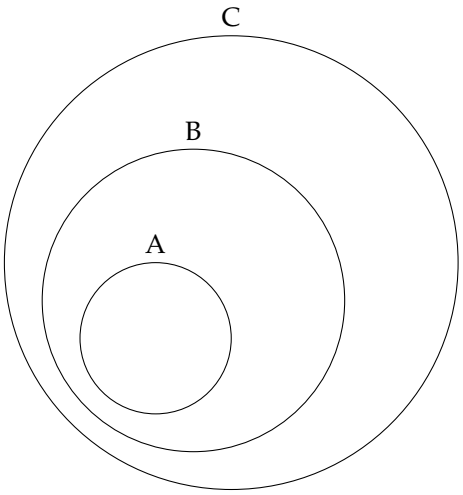
Resposta:

(prem, concl)	Argumento Válido	Argumento Inválido
(Verd, Verd)	Os pares são inteiros ∴ Os pares são inteiros	Os pares são múltiplos de 2 ∴ Os múltiplos de 2 são pares
(Verd, Falsa)	**Impossível**	Os inteiros são números ∴ Os números são inteiros
(Falsas, Verd)	As estrelas são pessoas As pessoas irradiam luz ∴ As estrelas irradiam luz	Os astros são estrelas ∴ As estrelas são astros
(Falsas, Falsa)	As bicicletas são casas ∴ As bicicletas são casas	Os patins são animais ∴ Os animais são patins

Uma maneira mais "automática" de resolver este exercício é considerar o teorema da forma, que diz que todos os argumentos com a mesma forma são válidos ou inválidos. Com base numa forma, encontrar os exemplos que satisfaçam o valor de verdade das premissas e conclusão.

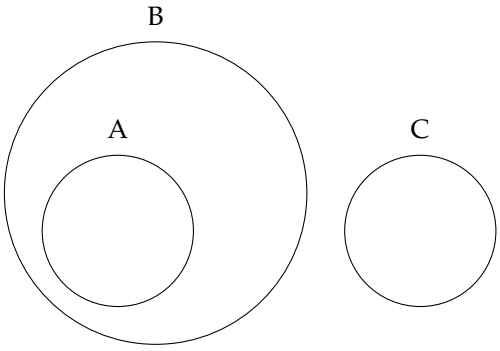
Para os argumentos válidos, sabemos que um argumento com a seguinte forma é válido:

Os As são Bs
Os Bs são Cs
∴ Os As são Cs



Para os argumentos inválidos, sabemos que um argumento com a seguinte forma não é válido:

Os As são Bs
∴ Os Cs são Bs



Agora, só é preciso encontrar exemplos de proposições com o valor de verdade pedido, de maneira a que o argumento satisfaça estas formas.

(prem, concl)	Argumento Válido	Argumento Inválido
(Verd, Verd)	Os pares são inteiros Os inteiros são números ∴ Os pares são números	Os inteiros são números ∴ Os reais são números
(Verd, Falsa)	**Impossível**	Os inteiros são números ∴ Os animais são números
(Falsas, Verd)	Os inteiros são pessoas As pessoas são números ∴ Os inteiros são números	Os animais são números ∴ Os inteiros são números
(Falsas, Falsa)	Os inteiros são pessoas As pessoas são plantas ∴ Os inteiros são plantas	Os animais são números ∴ Os carros são números

2 Lógica Proposicional — Sistema de Dedução Natural

Sumário:

- Regras do sistema de dedução natural da lógica proposicional
- Provas usando o sistema de dedução natural da lógica proposicional

Resumo:

As fórmulas bem formadas (ou fbfs) correspondem ao menor conjunto definido através das seguintes regras de formação:

1. Os símbolos de proposição são fbfs (chamadas fbfs atómicas);
2. Se α é uma fbf, então $(\neg\alpha)$ é uma fbf;
3. Se α e β são fbfs, então $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$ e $(\alpha \rightarrow \beta)$ são fbfs.

Uma *prova* (também chamada demonstração) é uma sequência finita de linhas numeradas, cada uma das quais ou contém uma premissa ou uma fbf que é adicionada à prova recorrendo a uma das regras de inferência e utilizando as fbfs que existem nas linhas anteriores. No lado direito de cada linha existe uma justificação da introdução da linha na prova.

Uma *prova de α a partir de Δ* é uma prova cuja última linha contém a fbf α e cujas restantes linhas contêm ou uma fbf em Δ ou uma fbf obtida a partir das fbfs das linhas anteriores através da aplicação de uma regra de inferência. Se existir uma prova de α a partir de Δ, dizemos que α é derivável a partir de Δ e escrevemos $\Delta \vdash \alpha$.

Existem dois tipos de provas: as provas *categóricas* que não são iniciadas com a introdução de uma hipótese; e as provas *hipotéticas* que são iniciadas com a introdução de uma hipótese. As fbfs de uma prova hipotética são chamadas *contingentes*, uma vez que dependem da hipótese que iniciou a prova; as fbfs nas provas exteriores são chamadas *categóricas*.

Uma fbf que é obtida numa prova que não contém premissas é chamada um *teorema*. Quando $\{\} \vdash \alpha$, ou seja, quando α é um teorema, é usual escrever apenas $\vdash \alpha$.

Os argumentos continuam a ser representados por um par (Δ, α). Quando uma fórmula α é derivável (usando o sistema sintáctico) a partir de um conjunto de fórmulas Δ, escrevemos $\Delta \vdash \alpha$. Quando uma fórmula α é consequência lógica (usando o sistema semântico) de um conjunto de fórmulas Δ, escrevemos $\Delta \models \alpha$.

Provas nos testes

Embora no livro sejam referidas regras de inferência derivadas, nos testes em geral o enunciado diz que apenas se podem usar as regras básicas: premissa, hipótese, repetição, re-iteração e introdução e eliminação de cada conectiva lógica.

Para simplificar as provas, permitimos que se usem algumas das regras básicas com mais graus de liberdade, mas sempre usando fbfs que estejam "no nível de prova correcto":

Em geral podemos não ter as fbfs pela ordem em que aparecem no enunciado da regra (uma vez que isso poderia ser obtido através da utilização de repetições), desde que os números dos passos estejam pela ordem correcta na justificação do passo correspondente.

$Reit$ podemos passar para dentro de mais do que um nível de cada vez.

Em todas as outras regras temos que respeitar os níveis de prova.

No fim dos exercícios está um resumo das regras de inferência básicas do sistema de dedução natural da lógica proposicional.

Exercício 2.1
(JPM) Demonstre os seguintes teoremas e argumentos usando o sistema de dedução natural da lógica proposicional. Em cada alínea indique se está a demonstrar um teorema ou um argumento.

1. $A \to (B \to A)$
2. $(A \land \neg A) \to B$
3. $(\{A \to B, B \to \neg A\}, \neg A)$
4. $(\{A\}, B \to (A \land B))$
5. $(\{\}, ((A \to B) \land (B \to C)) \to (A \to C))$

Resposta:

1. $A \to (B \to A)$ — teorema

1	A	Hip
2	$\quad B$	Hip
3	$\quad A$	Rei, 1
4	$B \to A$	I$\to$, (2, 3)
5	$A \to (B \to A)$	I$\to$, (1, 4)

2. $(A \land \neg A) \to B$ — teorema

1	$A \land \neg A$	Hip
2	$\quad \neg B$	Hip
3	$\quad A \land \neg A$	Rei, 1
4	$\quad A$	E$\land$, 3
5	$\quad \neg A$	E$\land$, 3
6	$\neg\neg B$	I$\neg$, (2, (4, 5))
7	B	E$\neg$, 6
8	$(A \land \neg A) \to B$	I$\to$, (1, 7)

3. $(\{A \to B, B \to \neg A\}, \neg A)$ — argumento

1	$A \to B$	Prem
2	$B \to \neg A$	Prem
3	$\quad A$	Hip
4	$\quad A \to B$	Rei, 1
5	$\quad B$	E$\to$, (3, 4)
6	$\quad B \to \neg A$	Rei, 2
7	$\quad \neg A$	E$\to$, (5, 6)
8	$\neg A$	I$\neg$, (3, (3, 7))

4. $(\{A\}, B \to (A \wedge B))$ — argumento

1	A	Prem
2	$\quad B$	Hip
3	$\quad A$	Rei, 1
4	$\quad A \wedge B$	I$\wedge$, (3, 2)
5	$B \to (A \wedge B)$	I$\to$, (2, 4)

5. $(\{\}, ((A \to B) \wedge (B \to C)) \to (A \to C))$ — argumento. Para além disso, e uma vez que o conjunto de premissas do argumento é vazio, a fbf que é a sua conclusão é um teorema.

1	$(A \to B) \wedge (B \to C)$	Hip
2	$A \to B$	E$\wedge$, 1
3	$B \to C$	E$\wedge$, 1
4	$\quad A$	Hip
5	$\quad A \to B$	Rei, 2
6	$\quad B$	E$\to$, (4, 5)
7	$\quad B \to C$	Rei, 3
8	$\quad C$	E$\to$, (6, 7)
9	$A \to C$	I$\to$, (4, 8)
10	$((A \to B) \wedge (B \to C)) \to (A \to C)$	I$\to$, (1, 9)

Exercício 2.2
Demonstre os seguintes teoremas usando o sistema de dedução natural da lógica proposicional.

1. $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$
2. $((A \to (A \to B)) \wedge A) \to B$
3. $(A \to B) \to (\neg B \to \neg A)$
4. $(\neg B \to \neg A) \to (A \to B)$
5. $(A \wedge B) \to \neg(A \to \neg B)$
6. $((A \to B) \wedge \neg B) \to \neg A$
7. $(A \to \neg A) \to \neg A$
8. $(A \vee B) \to (B \vee A)$
9. $((A \vee B) \vee C) \to (A \vee (B \vee C))$
10. $(A \wedge (B \vee C)) \to ((A \wedge B) \vee (A \wedge C))$
11. $((A \wedge B) \vee (A \wedge C)) \to (A \wedge (B \vee C))$

12. $((A \wedge B) \vee A) \to A$

Resposta:

1. $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$

1	$A \to (B \to C)$	Hip
2	$\quad A \to B$	Hip
3	$\quad\quad A$	Hip
4	$\quad\quad A \to B$	Rei, 2
5	$\quad\quad B$	E$\to$, (3, 4)
6	$\quad\quad A \to (B \to C)$	Rei, 1
7	$\quad\quad B \to C$	E$\to$, (3, 6)
8	$\quad\quad C$	E$\to$, (5, 7)
9	$\quad A \to C$	I$\to$, (3, 8)
10	$(A \to B) \to (A \to C)$	I$\to$, (2, 9)
11	$(A \to (B \to C)) \to ((A \to B) \to (A \to C))$	I$\to$, (1, 10)

2. $((A \to (A \to B)) \wedge A) \to B$

1	$(A \to (A \to B)) \wedge A$	Hip
2	$A \to (A \to B)$	E$\wedge$, 1
3	A	E$\wedge$, 1
4	$A \to B$	E$\to$, (3, 2)
5	B	E$\to$, (3, 4)
6	$((A \to (A \to B)) \wedge A) \to B$	I$\to$, (1, 5)

3. $(A \to B) \to (\neg B \to \neg A)$

1	$A \to B$	Hip
2	$\quad \neg B$	Hip
3	$\quad\quad A$	Hip
4	$\quad\quad A \to B$	Rei, 1
5	$\quad\quad B$	E$\to$, (3, 4)
6	$\quad\quad \neg B$	Rei, 2
7	$\quad \neg A$	I$\neg$, (3, (5, 6))
8	$\neg B \to \neg A$	I$\to$, (2, 7)
9	$(A \to B) \to (\neg B \to \neg A)$	I$\to$, (1, 8)

4. $(\neg B \to \neg A) \to (A \to B)$

1	$\neg B \to \neg A$	Hip
2	$\quad A$	Hip
3	$\quad\quad \neg B$	Hip
4	$\quad\quad \neg B \to \neg A$	Rei, 1
5	$\quad\quad \neg A$	E$\to$, (3, 4)
6	$\quad\quad A$	Rei, 2
7	$\quad \neg\neg B$	I$\neg$, (3, (6, 5))
8	$\quad B$	E$\neg$, 7
9	$A \to B$	I$\to$, (2, 8)
10	$(\neg B \to \neg A) \to (A \to B)$	I$\to$, (1, 9)

5. $(A \wedge B) \to \neg(A \to \neg B)$

1	$A \wedge B$	Hip
2	$\quad A \to \neg B$	Hip
3	$\quad A \wedge B$	Rei, 1
4	$\quad A$	E$\wedge$, 3
5	$\quad \neg B$	E$\to$, (4, 2)
6	$\quad B$	E$\wedge$, 3
7	$\neg(A \to \neg B)$	I$\neg$, (2, (6, 5))
8	$(A \wedge B) \to \neg(A \to \neg B)$	I$\to$, (1, 7)

6. $((A \to B) \wedge \neg B) \to \neg A$

1	$(A \to B) \wedge \neg B$	Hip
2	$\quad A$	Hip
3	$\quad (A \to B) \wedge \neg B$	Rei, 1
4	$\quad A \to B$	E$\wedge$, 3
5	$\quad B$	E$\to$, (2, 4)
6	$\quad \neg B$	E$\wedge$, 3
7	$\neg A$	I$\neg$, (2, (5, 6))
8	$((A \to B) \wedge \neg B) \to \neg A$	I$\to$, (1, 7)

7. $(A \to \neg A) \to \neg A$

1	$A \to \neg A$	Hip
2	$\quad A$	Hip
3	$\quad A \to \neg A$	Rei, 1
4	$\quad \neg A$	E$\to$, (2, 3)
5	$\neg A$	I$\neg$, (2, (2, 4))
6	$(A \to \neg A) \to \neg A$	I$\to$, (1, 5)

8. $(A \vee B) \to (B \vee A)$

1	$A \vee B$	Hip
2	$\quad A$	Hip
3	$\quad B \vee A$	I$\vee$, 2
4	$\quad B$	Hip
5	$\quad B \vee A$	I$\vee$, 4
6	$B \vee A$	E$\vee$, (1, (2, 3), (4, 5))
7	$(A \vee B) \to (B \vee A)$	I$\to$, (1, 6)

9. $((A \vee B) \vee C) \to (A \vee (B \vee C))$

1	$(A \vee B) \vee C$	Hip
2	$\quad A \vee B$	Hip
3	$\quad\quad A$	Hip
4	$\quad\quad A \vee (B \vee C)$	I$\vee$, 3
5	$\quad\quad B$	Hip
6	$\quad\quad B \vee C$	I$\vee$, 5
7	$\quad\quad A \vee (B \vee C)$	I$\vee$, 6
8	$\quad A \vee (B \vee C)$	E$\vee$, (2, (3, 4), (5, 7))
9	$\quad C$	Hip
10	$\quad B \vee C$	I$\vee$, 9
11	$\quad A \vee (B \vee C)$	I$\vee$, 10
12	$A \vee (B \vee C)$	E$\vee$, (1, (2, 8), (9, 11))
13	$((A \vee B) \vee C) \to (A \vee (B \vee C))$	I$\to$, (1, 12)

10. $(A \wedge (B \vee C)) \rightarrow ((A \wedge B) \vee (A \wedge C))$

1	$A \wedge (B \vee C)$	Hip
2	A	E$\wedge$, 1
3	$B \vee C$	E$\wedge$, 1
4	$\quad B$	Hip
5	$\quad A$	Rei, 2
6	$\quad A \wedge B$	I$\wedge$, (5, 4)
7	$\quad (A \wedge B) \vee (A \wedge C)$	I$\vee$, 6
8	$\quad C$	Hip
9	$\quad A$	Rei, 2
10	$\quad A \wedge C$	I$\wedge$, (9, 8)
11	$\quad (A \wedge B) \vee (A \wedge C)$	I$\vee$, 10
12	$(A \wedge B) \vee (A \wedge C)$	E$\vee$, (1, (4, 7), (8, 11))
13	$(A \wedge (B \vee C)) \rightarrow ((A \wedge B) \vee (A \wedge C))$	I$\rightarrow$, (1, 12)

11. $((A \wedge B) \vee (A \wedge C)) \rightarrow (A \wedge (B \vee C))$

1	$(A \wedge B) \vee (A \wedge C)$	Hip
2	$\quad A \wedge B$	Hip
3	$\quad A$	E$\wedge$, 2
4	$\quad B$	E$\wedge$, 2
5	$\quad B \vee C$	I$\vee$, 4
6	$\quad A \wedge (B \vee C)$	I$\wedge$, (3, 5)
7	$\quad A \wedge C$	Hip
8	$\quad A$	E$\wedge$, 7
9	$\quad C$	E$\wedge$, 7
10	$\quad B \vee C$	I$\vee$, 9
11	$\quad A \wedge (B \vee C)$	I$\wedge$, (8, 10)
12	$A \wedge (B \vee C)$	E$\vee$, (1, (2, 6), (7, 11))
13	$((A \wedge B) \vee (A \wedge C)) \rightarrow (A \wedge (B \vee C))$	I$\rightarrow$, (1, 12)

12. $((A \wedge B) \vee A) \rightarrow A$

1	$(A \wedge B) \vee A$	Hip
2	$\quad A \wedge B$	Hip
3	$\quad A$	E$\wedge$, 2
4	$\quad A$	Hip
5	$\quad A$	Rep, 4
6	A	E$\vee$, (1, (2, 3), (4, 5))
7	$((A \wedge B) \vee A) \rightarrow A$	I$\rightarrow$, (1, 6)

Exercício 2.3
Demonstre os seguintes teoremas, que correspondem a aplicações das leis de De Morgan, usando o sistema de dedução natural da lógica proposicional.

1. $\neg(A \vee B) \rightarrow (\neg A \wedge \neg B)$
2. $(\neg A \wedge \neg B) \rightarrow \neg(A \vee B)$
3. $\neg(A \wedge B) \rightarrow (\neg A \vee \neg B)$
4. $(\neg A \vee \neg B) \rightarrow \neg(A \wedge B)$

Resposta:

1. $\neg(A \vee B) \rightarrow (\neg A \wedge \neg B)$

1	$\neg(A \vee B)$	Hip
2	$\quad A$	Hip
3	$\quad A \vee B$	I$\vee$, 2
4	$\quad \neg(A \vee B)$	Rei, 1
5	$\neg A$	I$\neg$, (2, (3, 4))
6	$\quad B$	Hip
7	$\quad A \vee B$	I$\vee$, 6
8	$\quad \neg(A \vee B)$	Rei, 1
9	$\neg B$	I$\neg$, (6, (7, 8))
10	$\neg A \wedge \neg B$	I$\wedge$, (5, 9)
11	$\neg(A \vee B) \rightarrow (\neg A \wedge \neg B)$	I$\rightarrow$, (1, 10)

2. $(\neg A \wedge \neg B) \rightarrow \neg(A \vee B)$

ou

1	$\neg A \wedge \neg B$	Hip
2	$\neg A$	E$\wedge$, 1
3	$\neg B$	E$\wedge$, 1
4	$\quad A \vee B$	Hip
5	$\quad\quad A$	Hip
6	$\quad\quad\quad \neg A \wedge \neg B$	Hip
7	$\quad\quad\quad A$	Rei, 5
8	$\quad\quad\quad \neg A$	Rei, 2
9	$\quad\quad \neg(\neg A \wedge \neg B)$	I$\neg$, (6, (7, 8))
10	$\quad\quad B$	Hip
11	$\quad\quad\quad \neg A \wedge \neg B$	Hip
12	$\quad\quad\quad B$	Rei, 10
13	$\quad\quad\quad \neg B$	Rei, 3
14	$\quad\quad \neg(\neg A \wedge \neg B)$	I$\neg$, (11, (12, 13))
15	$\quad \neg(\neg A \wedge \neg B)$	E$\vee$, (4, (5, 9), (10, 14))
16	$\quad \neg A \wedge \neg B$	Rei, 1
17	$\neg(A \vee B)$	I$\neg$, (4, (16, 15))
18	$(\neg A \wedge \neg B) \rightarrow \neg(A \vee B)$	I$\rightarrow$, (1, 17)

1	$\neg A \wedge \neg B$	Hip
2	$\neg A$	E$\wedge$, 1
3	$\neg B$	E$\wedge$, 1
4	$\quad A \vee B$	Hip
5	$\quad\quad A$	Hip
6	$\quad\quad A$	Rep, 5
7	$\quad\quad B$	Hip
8	$\quad\quad\quad \neg A$	Hip
9	$\quad\quad\quad \neg B$	Rei, 3
10	$\quad\quad\quad B$	Rei, 7
11	$\quad\quad \neg\neg A$	I$\neg$, (8, (10, 9))
12	$\quad\quad A$	E$\neg$, 11
13	$\quad A$	E$\vee$, (4, (5, 6), (7, 12))
14	$\quad \neg A$	Rei, 2
15	$\neg(A \vee B)$	I$\neg$, (4, (13, 14))
16	$(\neg A \wedge \neg B) \rightarrow \neg(A \vee B)$	I$\rightarrow$, (1, 15)

3. $\neg(A \wedge B) \to (\neg A \vee \neg B)$

1	$\neg(A \wedge B)$	Hip
2	$\quad \neg(\neg A \vee \neg B)$	Hip
3	$\quad\quad \neg A$	Hip
4	$\quad\quad \neg A \vee \neg B$	I$\vee$, 3
5	$\quad\quad \neg(\neg A \vee \neg B)$	Rei, 2
6	$\quad \neg\neg A$	I$\neg$, (3, (4, 5))
7	$\quad A$	E$\neg$, 6
8	$\quad\quad \neg B$	Hip
9	$\quad\quad \neg A \vee \neg B$	I$\vee$, 8
10	$\quad\quad \neg(\neg A \vee \neg B)$	Rei, 2
11	$\quad \neg\neg B$	I$\neg$, (8, (9, 10))
12	$\quad B$	E$\neg$, 11
13	$\quad A \wedge B$	I$\wedge$, (7, 12)
14	$\quad \neg(A \wedge B)$	Rei, 1
15	$\neg\neg(\neg A \vee \neg B)$	I$\neg$, (2, (13, 14))
16	$\neg A \vee \neg B$	E$\neg$, 15
17	$\neg(A \wedge B) \to (\neg A \vee \neg B)$	I$\to$, (1, 16)

4. $(\neg A \vee \neg B) \to \neg(A \wedge B)$

ou

1	$\neg A \vee \neg B$	Hip
2	$\quad \neg A$	Hip
3	$\quad\quad A \wedge B$	Hip
4	$\quad\quad A$	E$\wedge$, 3
5	$\quad\quad \neg A$	Rei, 2
6	$\quad \neg(A \wedge B)$	I$\neg$, (3, (4, 5))
7	$\quad \neg B$	Hip
8	$\quad\quad A \wedge B$	Hip
9	$\quad\quad B$	E$\wedge$, 8
10	$\quad\quad \neg B$	Rei, 7
11	$\quad \neg(A \wedge B)$	I$\neg$, (8, (9, 10))
12	$\neg(A \wedge B)$	E$\vee$, (1, (2, 6), (7, 11))
13	$(\neg A \vee \neg B) \rightarrow \neg(A \wedge B)$	I$\rightarrow$, (1, 12)

1	$\neg A \vee \neg B$	Hip
2	$\quad A \wedge B$	Hip
3	$\quad A$	E$\wedge$, 2
4	$\quad B$	E$\wedge$, 2
5	$\quad \neg A \vee \neg B$	Rei, 1
6	$\quad\quad \neg A$	Hip
7	$\quad\quad\quad B$	Hip
8	$\quad\quad\quad A$	Rei, 3
9	$\quad\quad\quad \neg A$	Rei, 6
10	$\quad\quad \neg B$	I$\neg$, (7, (8, 9))
11	$\quad\quad \neg B$	Hip
12	$\quad\quad \neg B$	Rep, 11
13	$\quad \neg B$	E$\vee$, (5, (6, 10), (11, 12))
14	$\neg(A \wedge B)$	I$\neg$, (2, (4, 13))
15	$(\neg A \vee \neg B) \rightarrow \neg(A \wedge B)$	I$\rightarrow$, (1, 14)

Lógica Proposicional — Sistema de Dedução Natural

Prem

$n \quad \alpha \qquad$ Prem

Hip

$n \quad \alpha \qquad$ Hip

$n+1 \quad \ldots$

Rep

$n \quad \alpha$

$\vdots \quad \vdots$

$m \quad \alpha \qquad$ Rep, n

Rei

$n \quad \alpha$

$\vdots \quad \vdots$

$m \quad \alpha \qquad$ Rei, n

I $\rightarrow$

$n \quad \alpha \qquad$ Hip

$\vdots \quad \vdots$

$m \quad \beta$

$k \quad \alpha \rightarrow \beta \qquad$ I$\rightarrow$, (n, m)

E $\rightarrow$

$n \quad \alpha$

$\vdots \quad \vdots$

$m \quad \alpha \rightarrow \beta$

$\vdots \quad \vdots$

$k \quad \beta \qquad$ E$\rightarrow$, (n, m)

I$\wedge$

$n \quad \alpha$

$\vdots \quad \vdots$

$m \quad \beta$

$\vdots \quad \vdots$

$k \quad \alpha \wedge \beta \qquad$ I$\wedge$, (n, m)

E$\wedge$

$n \quad \alpha \wedge \beta$

$\vdots \quad \vdots$

$m \quad \alpha \qquad$ E$\wedge$, n

ou

$n \quad \alpha \wedge \beta$

$\vdots \quad \vdots$

$m \quad \beta \qquad$ E$\wedge$, n

I$\vee$

$n \quad \alpha$

$\vdots \quad \vdots$

$m \quad \alpha \vee \beta \qquad$ I$\vee$, n

ou

$n \quad \alpha$

$\vdots \quad \vdots$

$m \quad \beta \vee \alpha \qquad$ I$\vee$, n

E$\vee$

$n \quad \alpha \vee \beta$

$o \quad \alpha \qquad$ Hip

$\vdots \quad \vdots$

$p \quad \gamma$

$r \quad \beta \qquad$ Hip

$\vdots \quad \vdots$

$s \quad \gamma$

$m \quad \gamma \qquad$ E$\vee$, $(n, (o, p), (r, s))$

I$\neg$

$n \quad \alpha \qquad$ Hip

$\vdots \quad \vdots$

$m \quad \beta$

$\vdots \quad \vdots$

$k \quad \neg\beta$

$l \quad \neg\alpha \qquad$ I$\neg$, $(n, (m, k))$

E$\neg$

$n \quad \neg\neg\alpha$

$\vdots \quad \vdots$

$m \quad \alpha \qquad$ E$\neg$, n

3 Lógica Proposicional — Tabelas de Verdade

Sumário:

- Provas usando o sistema semântico da lógica proposicional
- Interpretações
- Tabelas de verdade

Resumo:

Uma *função de valoração* é uma função dos símbolos de proposição para os valores lógicos, $verdadeiro$ e $falso$.

Dada uma função de valoração, uma *interpretação* é uma função das fbfs para os valores lógicos, que atribui às frases da linguagem o valor $verdadeiro$ ou $falso$.

Dado um símbolo de proposição P e uma interpretação v, dizemos que v *satisfaz* o símbolo de proposição P sse $v(P) = V$ (diz-se também que o símbolo de proposição P é verdadeiro segundo a interpretação v); caso contrário, dizemos que a interpretação não satisfaz o símbolo de proposição P (neste caso, diz-se também que o símbolo de proposição P é falso segundo a interpretação v).

Um *modelo* de uma fbf α é uma interpretação que satisfaz α. Um *modelo* de um conjunto de fbfs Δ é uma interpretação que satisfaz todas as fbfs de Δ.

Diz-se que uma fbf α é *consequência lógica* de um conjunto de fbfs Δ quando for impossível encontrar uma interpretação que satisfaça todas as fbfs do conjunto Δ mas não satisfaça α. Ou seja, quando α for satisfeita por todos os modelos do conjunto Δ.

As *tabelas de verdade* para as fbfs são construidas tendo em conta as valorações dos símbolos de proposição e as tabelas de verdade das conectivas lógicas, já conhecidas.

Usando a semântica, um argumento é *válido* se não existir nenhuma interpretação que torne todas as premissas verdadeiras e a conclusão falsa, ou seja, se não existir nenhuma linha da tabela de verdade em que todas as premissas tenham o valor V e a conclusão tenha o valor F.

Uma fbf diz-se *satisfazível* sse existe pelo menos uma interpretação na qual a fbf é verdadeira.

Uma fbf diz-se *falsificável* sse existe pelo menos uma interpretação na qual a fbf é falsa.

Uma fbf diz-se *tautológica* sse é verdadeira para todas as interpretações.

Uma fbf diz-se *contraditória* sse é falsa para todas as interpretações.

A lógica proposicional é correcta e completa. Isto é importante para nos permitir fazer provas pela via sintáctica, com base no conhecimento que temos, e sabermos que isso tem uma "correspondência semântica" no mundo que estamos a representar. Enquanto que na lógica proposicional é relativamente simples encontrar algoritmos eficientes para fazer provas pela via semântica, na lógica de primeira ordem isso é mais complicado.

Assim, fazemos as provas pela via sintáctica e, pela correcção e completude da lógica, sabemos que seríamos capazes de provar o mesmo pela via semântica.

Uma lógica é *correcta* (do inglês, "sound") se qualquer argumento demonstrável (com o seu sistema dedutivo) é válido de acordo com a sua semântica. Ou seja, se partindo de um conjunto de premissas, o sistema dedutivo não gera nenhuma conclusão errada.

Uma lógica é *completa* (do inglês, "complete") se qualquer argumento válido de acordo com a sua semântica é demonstrável no seu sistema dedutivo. Ou seja, significa que podemos provar todas as proposições verdadeiras, tendo em atenção as premissas.

Exercício 3.1
Para cada uma das seguintes fbfs, diga, justificando, se é satisfazível, falsificável, tautológica ou contraditória. É necessário conhecer todas as linhas das tabelas de verdade de cada fbf para responder a estas perguntas?

1. A
2. $A \wedge \neg A$
3. $A \vee \neg A$
4. $(A \wedge B) \rightarrow (A \vee B)$
5. $(A \vee B) \rightarrow (A \wedge B)$

Resposta:

Fórmula	**Satisfazível** Se tiver pelo menos uma interpretação Verdadeira.	**Falsificável** Se tiver pelo menos uma interpretação Falsa.	**Tautológica** Se tiver todas as interpretações Verdadeiras.	**Contraditória** Se tiver todas as interpretações Falsas.
A	Sim	Sim	Não	Não
$A \wedge \neg A$	Não	Sim	Não	Sim
$A \vee \neg A$	Sim	Não	Sim	Não
$(A \wedge B) \rightarrow (A \vee B)$	Sim	Não	Sim	Não
$(A \vee B) \rightarrow (A \wedge B)$	Sim	Sim	Não	Não

Para provar que uma fórmula é satisfazível, podemos parar de procurar assim que encontrarmos uma interpretação que satisfaça a fórmula, por isso pode não ser necessário conhecer toda a tabela de verdade. Só será necessário conhecer toda a tabela no caso de a fórmula não ser satisfazível. Podemos dizer algo semelhante acerca das fórmulas falsificáveis, mas paramos quando encontrarmos uma interpretação que não satisfaça a fórmula.

Para provar que uma fórmula é tautológica podemos parar numa de duas situações: ou paramos quando encontramos uma interpretação que não satisfaça a fórmula e neste caso não é tautológica, ou então temos que mostrar que todas as interpretações satisfazem a fórmula e neste caso precisamos de percorrer toda a tabela. Podemos dizer algo semelhante para as fórmulas contraditórias, trocando "satisfazer" por "não satisfazer" e vice-versa.

Exercício 3.2
Considere o seguinte conjunto de fórmulas:

$$\{Homem \rightarrow Pessoa, Mulher \rightarrow Pessoa, Homem \vee Mulher\}$$

1. Mostre quais são os modelos desse conjunto.
2. $Pessoa$ é consequência lógica desse conjunto? Porquê?
3. Acrescente $\neg Homem$ ao conjunto. Diga quais são os seus modelos e as suas consequências lógicas.

Resposta:

Considerando que: H representa $Homem$, M representa $Mulher$ e P representa $Pessoa$.

1.

H	M	P	$H \to P$	$M \to P$	$H \vee M$	É modelo?
V	V	V	V	V	V	S
V	V	F	F	F	V	N
V	F	V	V	V	V	S
V	F	F	F	V	V	N
F	V	V	V	V	V	S
F	V	F	V	F	V	N
F	F	V	V	V	F	N
F	F	F	V	V	F	N

Um modelo de um conjunto de fórmulas é uma interpretação que satisfaz todas as fbfs desse conjunto, isto é, que as torna todas verdadeiras. Assim, os modelos deste conjunto de fórmulas correspondem às interpretações que atribuem a $Homem$, $Mulher$ e $Pessoa$ os valores das colunas respectivas, nas linhas em que o valor da última coluna é "S".

Convém notar que num dos modelos temos como consequência lógica $Homem \wedge Mulher$, pois não foi dito nada que o impedisse.

2. As consequências lógicas de um conjunto de fórmulas são as fórmulas que têm o valor V em qualquer modelo do conjunto. Assim, $Pessoa$ é consequência lógica deste conjunto, porque tem o valor V em todos os modelos do conjunto.

   Intuitivamente, uma vez que temos $Homem \vee Mulher$, faz sentido que $Pessoa$ seja consequência lógica do conjunto.

3. Para este novo conjunto, só interessa a metade inferior da tabela, em que $Homem$ tem o valor F. O modelo é o que está assinalado na coluna à direita. As consequências lógicas deste novo conjunto são infinitas e incluem:

   - As fórmulas que estão no conjunto: $\neg Homem$, $Homem \to Pessoa$, $Mulher \to Pessoa$, $Homem \vee Mulher$,
   - as fórmulas atómicas às quais a interpretação atribui o valor verdadeiro: $Mulher$ e $Pessoa$,
   - todas as fórmulas deriváveis a partir das fórmulas anteriores: $Mulher \wedge Pessoa$, $Mulher \vee Boneca$, $Homem \to Mulher$, etc.,
   - todos os teoremas da lógica proposicional: $A \to (B \to A)$, $(A \wedge \neg A) \to B$, $A \vee \neg A$, etc.

   Nota: se a pergunta fosse as fórmulas *deriváveis*, ainda seria necessário fazer as provas respectivas pela via sintáctica.

Exercício 3.3
Considere a seguinte tabela de verdade. Diga a que conectiva lógica corresponde a função f.

P	Q	$f(P,Q)$
V	V	V
V	F	F
F	V	F
F	F	F

Resposta:

A função f corresponde a um "e" lógico ou conjunção.

Exercício 3.4
Usando o sistema semântico da lógica proposicional, mostre que $\{\neg A, \neg B\} \models \neg(A \vee B)$.

Resposta:

Para uma interpretação satisfazer $\{\neg A, \neg B\}$, A e B têm que ser ambos falsos. Assim, $(A \vee B)$ também será falso e por isso $\neg(A \vee B)$ será verdadeiro. Logo, todas as interpretações que satisfazem $\{\neg A, \neg B\}$ também satisfazem $\neg(A \vee B)$, ou seja, $\{\neg A, \neg B\} \models \neg(A \vee B)$.

Outra possibilidade de resposta seria fazer uma tabela de verdade e mostrar que todas as linhas da tabela (interpretações) que tornam simultaneamente $\neg A$ e $\neg B$ verdadeiras também tornam a fbf $\neg(A \vee B)$ verdadeira.

Exercício 3.5
Mostre que as fórmulas $A \rightarrow (B \rightarrow C)$ e $(A \rightarrow B) \rightarrow C$ não são equivalentes apresentando uma interpretação para a qual elas tenham valores lógicos diferentes.

Resposta:

Um exemplo:
$I_1(A) = F$
$I_1(B) = F$
$I_1(C) = F$
Neste caso, temos $I_1(A \rightarrow (B \rightarrow C)) = V$ e $I_1((A \rightarrow B) \rightarrow C) = F$.
Outro exemplo:
$I_2(A) = F$
$I_2(B) = V$
$I_2(C) = F$
Neste caso, temos $I_2(A \rightarrow (B \rightarrow C)) = V$ e $I_2((A \rightarrow B) \rightarrow C) = F$.

Outra possibilidade de resposta seria fazer a tabela de verdade com as oito interpretações possíveis para A, B e C e mostrar que em pelo menos uma delas as duas fbfs do enunciado têm valores de verdade diferentes.

Exercício 3.6
Usando tabelas de verdade, prove que as seguintes fórmulas, correspondentes às leis de De Morgan, são tautologias.

$$\neg(A \vee B) \leftrightarrow (\neg A \wedge \neg B)$$
$$\neg(A \wedge B) \leftrightarrow (\neg A \vee \neg B)$$

Resposta:

A	B	$\neg(A \vee B)$	$\neg A \wedge \neg B$	$\neg(A \vee B) \leftrightarrow (\neg A \wedge \neg B)$
V	V	F	F	V
V	F	F	F	V
F	V	F	F	V
F	F	V	V	V

A	B	$\neg(A \wedge B)$	$\neg A \vee \neg B$	$\neg(A \wedge B) \leftrightarrow (\neg A \vee \neg B)$
V	V	F	F	V
V	F	V	V	V
F	V	V	V	V
F	F	V	V	V

Cada fórmula tem sempre o valor verdadeiro independentemente da valoração de A e de B, por isso são tautologias.

4 Lógica Proposicional — Resolução

Sumário:

- Resolução em lógica proposicional
- Provas usando resolução em lógica proposicional

Resumo:

A *resolução* surgiu da necessidade de criar um modo simples e eficiente de automatizar o raciocínio lógico, para poder ser facilmente implementado em computador.

Antes de se poder usar a *resolução*, as fbfs têm que ser passadas para a *forma clausal*, que corresponde a uma conjunção de cláusulas. Uma *cláusula* é um literal ou uma disjunção de literais. Ou seja, uma fbf na forma clausal corresponde a uma conjunção de disjunções.

Uma fbf atómica corresponde a um símbolo de proposição. Uma fbf atómica ou a sua negação é chamada um *literal*. Um *literal positivo* corresponde a uma fbf atómica e um *literal negativo* corresponde à negação de uma fbf atómica.

Passos da passagem para a forma clausal:

1. Eliminação de $\rightarrow$: substituir todas as ocorrências de $\alpha \rightarrow \beta$ por $\neg\alpha \vee \beta$ (normalmente, de dentro para fora).
2. Redução do domínio de $\neg$: eliminação da dupla negação $\neg\neg\alpha \leftrightarrow \alpha$, utilização das leis de De Morgan $\neg(\alpha \vee \beta) \leftrightarrow \neg\alpha \wedge \neg\beta$ e $\neg(\alpha \wedge \beta) \leftrightarrow \neg\alpha \vee \neg\beta$ (normalmente, de fora para dentro).
3. Obtenção da forma conjuntiva normal: transformar a fórmula numa conjunção de disjunções usando a propriedade distributiva da disjunção em relação à conjunção $\alpha \vee (\beta \wedge \gamma) \leftrightarrow (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$.
4. Eliminação de $\wedge$: transformar a fórmula num conjunto de cláusulas.
5. Eliminação de $\vee$: transformar cada cláusula num conjunto de literais.

Princípio da resolução: a partir de $\alpha \vee \beta$ e de $\neg\alpha \vee \gamma$, podemos concluir $\beta \vee \gamma$. Formalmente, sendo Ψ e Φ duas cláusulas e α uma fbf atómica tal que $\alpha \in \Psi$ e $\neg\alpha \in \Phi$, então, usando o princípio da resolução, podemos inferir a cláusula $(\Psi - \{\alpha\}) \cup (\Phi - \{\neg\alpha\})$, que é o *resolvente* das cláusulas Ψ e Φ.

1	$\{A\}$	Prem
2	$\{\neg A, B\}$	Prem
3	$\{B\}$	Res, (1, 2)

A resolução é correcta mas não é completa (por exemplo, $\{P\} \models \{P, Q\}$ mas não é possível derivar a conclusão $\{P, Q\}$ a partir da premissa $\{P\}$ usando resolução). No entanto, é completa quanto à refutação (pois é possível chegar a uma contradição, representada

por $\{\}$, a partir da premissa $\{P\}$ e das cláusulas correspondentes à negação da conclusão $\{\neg P\}, \{\neg Q\}$).

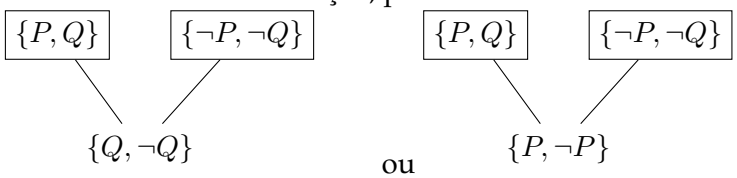
Normalmente, aplica-se a resolução a demonstrações por *redução ao absurdo*, em que o objectivo é chegar a uma cláusula vazia, ou seja, a uma contradição:

- Para provar que a conclusão de um argumento é derivável a partir das suas premissas, passam-se as premissas e a negação da conclusão para a forma clausal e tenta-se chegar à cláusula vazia.
- Para provar que uma fórmula é teorema, nega-se a fórmula, passa-se para a forma clausal, e tenta-se chegar a uma cláusula vazia. Note-se que isto é o mesmo que tentar provar um argumento em que o conjunto de premissas é vazio.
- Convém ter em atenção que todas as cláusulas resultantes da passagem para a forma clausal das premissas e da negação da conclusão do argumento vão ser premissas da resolução. A palavra "premissa" tem dois significados diferentes, é preciso decidir qual usar com base no contexto (se são as premissas do argumento ou as premissas da resolução).

Estratégias a usar em resolução:

- Geração por saturação de níveis: em cada nível usa-se pelo menos uma cláusula do nível anterior.
- Estratégias de eliminação de cláusulas eliminam cláusulas que não vão ser úteis numa prova por resolução: eliminar os teoremas (ou tautologias), eliminar as cláusulas não mínimas (ou subordinadas) e eliminar as cláusulas que contêm literais puros (cuja negação não aparece noutras cláusulas).
- Resolução unitária: utiliza sempre pelo menos uma cláusula com apenas um literal.
- Resolução linear: consiste em utilizar uma cláusula e os sucessores dessa cláusula sempre que se aplica o princípio da resolução. Ou seja, cada passo usa sempre o resultado do passo anterior.
- Resolução com conjunto de apoio: aplica-se normalmente a demonstrações por redução ao absurdo e tenta aplicar o princípio da resolução a pelo menos uma cláusula do conjunto de apoio. O conjunto de apoio corresponde à negação do que se pretende provar.

Convém lembrar que o princípio da resolução permite eliminar **uma** fbf atómica e a sua negação de cada vez. Por exemplo, dada a fbf $(P \vee Q) \wedge (\neg P \vee \neg Q)$, na forma clausal temos $\{\{P, Q\}, \{\neg P, \neg Q\}\}$ e poderíamos ser tentados a eliminar P e Q num passo só, ficando com a cláusula vazia. Mas isto estaria errado, pois usando a fbf inicial sabemos que apenas um de P e Q será verdadeiro, mas não sabemos qual, nem temos forma de descobrir. Usando resolução, podemos fazer



mas não conseguimos chegar a $\{\}$.

Exercício 4.1
Transforme a seguinte fórmula para a forma clausal:

1. $A \to (\neg(\neg B \vee C) \vee (C \to D))$

Resposta:

1. $A \to (\neg(\neg B \vee C) \vee (C \to D))$
   (a) Eliminação de $\to$: $\neg A \vee (\neg(\neg B \vee C) \vee (\neg C \vee D))$
   (b) Redução do domínio de $\neg$: $\neg A \vee ((\neg\neg B \wedge \neg C) \vee (\neg C \vee D))$
   $\neg A \vee ((B \wedge \neg C) \vee (\neg C \vee D))$
   (c) Obtenção da forma conjuntiva normal: $\neg A \vee ((B \vee (\neg C \vee D)) \wedge (\neg C \vee (\neg C \vee D)))$
   $(\neg A \vee (B \vee (\neg C \vee D))) \wedge (\neg A \vee (\neg C \vee (\neg C \vee D)))$
   $(\neg A \vee B \vee \neg C \vee D) \wedge (\neg A \vee \neg C \vee D)$
   (d) Eliminação de $\wedge$: $\{\neg A \vee B \vee \neg C \vee D, \neg A \vee \neg C \vee D\}$
   (e) Eliminação de $\vee$: $\{\{\neg A, B, \neg C, D\}, \{\neg A, \neg C, D\}\}$

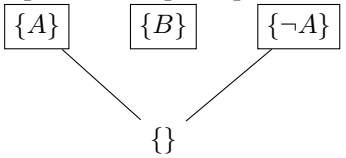
   Nota: neste exercício não se nega a fbf inicial, pois a única coisa que foi pedida foi que a passasse para a forma clausal, e não que se provasse alguma coisa com ela.

Exercício 4.2
Demonstre os teoremas e argumentos do exercício 2.1 usando resolução. Em cada alínea indique se está a demonstrar um teorema ou um argumento.

Resposta:

1. $A \to (B \to A)$ — teorema
   Estamos a tentar provar que uma fórmula é teorema. Negamo-la, passamo-la para a forma clausal, e tentamos chegar a uma cláusula vazia usando resolução.
   Passagem para a forma clausal de $\neg(A \to (B \to A))$:
   (a) Eliminação de $\to$: $\neg(\neg A \vee (\neg B \vee A))$
   (b) Redução do domínio de $\neg$: $\neg\neg A \wedge \neg(\neg B \vee A)$
   $A \wedge (\neg\neg B \wedge \neg A)$
   $A \wedge B \wedge \neg A$
   (c) Obtenção da forma conjuntiva normal: já está.
   (d) Eliminação de $\wedge$: $\{A, B, \neg A\}$
   (e) Eliminação de $\vee$: $\{\{A\}, \{B\}, \{\neg A\}\}$

   Aplicação do princípio da resolução:



   Nota1: Não foi necessário usar todas as premissas da resolução para chegar a uma cláusula vazia. Na realidade, isto faz sentido, uma vez que a contradição não depende de B, mas apenas de A e de $\neg A$.
   Nota2: Se não negarmos a fbf inicial e a passarmos directamente para a forma clausal, temos $A \to (B \to A)$, depois $\neg A \vee (\neg B \vee A)$, e finalmente $\{\{\neg A, \neg B, A\}\}$, e não podemos aplicar resolução a apenas uma cláusula, por isso nunca conseguiremos chegar à cláusula vazia. Não esquecer que em resolução as provas são feitas por refutação.

2. $(A \wedge \neg A) \rightarrow B$ — teorema
   Passagem para a forma clausal de $\neg((A \wedge \neg A) \rightarrow B)$:

   (a) Eliminação de $\rightarrow$: $\neg(\neg(A \wedge \neg A) \vee B)$

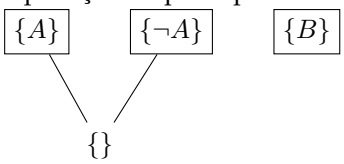
   (b) Redução do domínio de $\neg$: $\neg((\neg A \vee \neg\neg A) \vee B)$
   $\neg((\neg A \vee A) \vee B)$
   $\neg(\neg A \vee A) \wedge \neg B$
   $(A \wedge \neg A) \wedge \neg B$

   (c) Obtenção da forma conjuntiva normal: já está.

   (d) Eliminação de $\wedge$: $\{A, \neg A, B\}$

   (e) Eliminação de $\vee$: $\{\{A\}, \{\neg A\}, \{B\}\}$

   Aplicação do princípio da resolução:



3. $(\{A \rightarrow B, B \rightarrow \neg A\}, \neg A)$ — argumento
   Estamos a tentar provar um argumento. Cada premissa (individualmente) e a conclusão negada são passadas para a forma clausal e tenta-se chegar a uma contradição.

   Passagem para a forma clausal de $A \rightarrow B, B \rightarrow \neg A, \neg\neg A$:

   (a) Eliminação de $\rightarrow$: $\neg A \vee B, \neg B \vee \neg A, \neg\neg A$

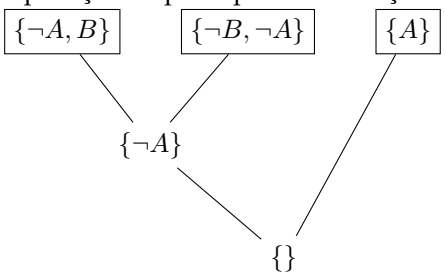
   (b) Redução do domínio de $\neg$: $\neg A \vee B, \neg B \vee \neg A, A$

   (c) Obtenção da forma conjuntiva normal: já está.

   (d) Eliminação de $\wedge$: $\{\neg A \vee B, \neg B \vee \neg A, A\}$

   (e) Eliminação de $\vee$: $\{\{\neg A, B\}, \{\neg B, \neg A\}, \{A\}\}$

   Aplicação do princípio da resolução:



4. $(\{A\}, B \rightarrow (A \wedge B))$ — argumento
   Passagem para a forma clausal de $A, \neg(B \rightarrow (A \wedge B))$:

   (a) Eliminação de $\rightarrow$: $A, \neg(\neg B \vee (A \wedge B))$

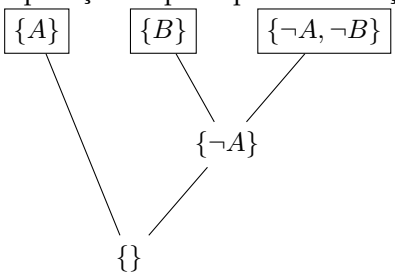
   (b) Redução do domínio de $\neg$: $A, \neg\neg B \wedge \neg(A \wedge B)$
   $A, B \wedge (\neg A \vee \neg B)$

   (c) Obtenção da forma conjuntiva normal: já está.

   (d) Eliminação de $\wedge$: $\{A, B, \neg A \vee \neg B\}$

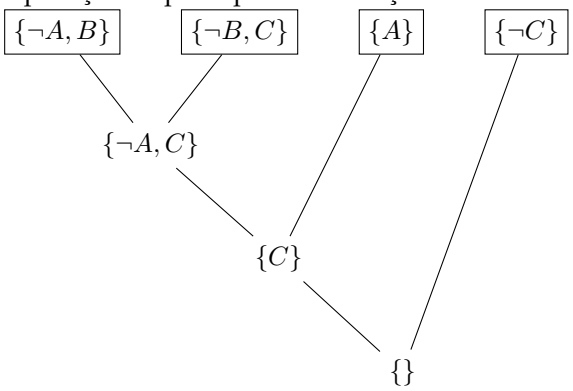
   (e) Eliminação de $\vee$: $\{\{A\}, \{B\}, \{\neg A, \neg B\}\}$

Aplicação do princípio da resolução:



5. $(\{\}, ((A \to B) \wedge (B \to C)) \to (A \to C))$ — argumento
   Passagem para a forma clausal de $\neg(((A \to B) \wedge (B \to C)) \to (A \to C))$:
   (a) Eliminação de $\to$: $\neg(\neg((\neg A \vee B) \wedge (\neg B \vee C)) \vee (\neg A \vee C))$
   (b) Redução do domínio de $\neg$: $\neg(\neg((\neg A \vee B) \wedge (\neg B \vee C)) \vee (\neg A \vee C))$
   $\neg((\neg(\neg A \vee B) \vee \neg(\neg B \vee C)) \vee (\neg A \vee C))$
   $\neg\neg(\neg A \vee B) \wedge \neg\neg(\neg B \vee C) \wedge \neg(\neg A \vee C)$
   $(\neg A \vee B) \wedge (\neg B \vee C) \wedge \neg\neg A \wedge \neg C$
   $(\neg A \vee B) \wedge (\neg B \vee C) \wedge A \wedge \neg C$
   (c) Obtenção da forma conjuntiva normal: já está.
   (d) Eliminação de $\wedge$: $\{\neg A \vee B, \neg B \vee C, A, \neg C\}$
   (e) Eliminação de $\vee$: $\{\{\neg A, B\}, \{\neg B, C\}, \{A\}, \{\neg C\}\}$

   Aplicação do princípio da resolução:

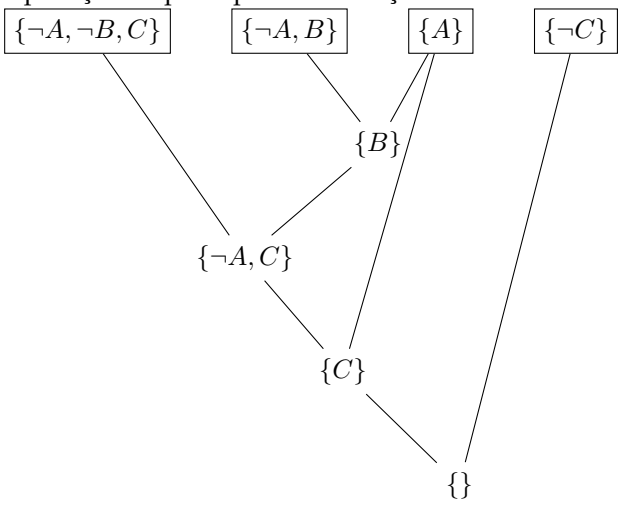


Exercício 4.3
Demonstre os teoremas do exercício 2.2 usando resolução.

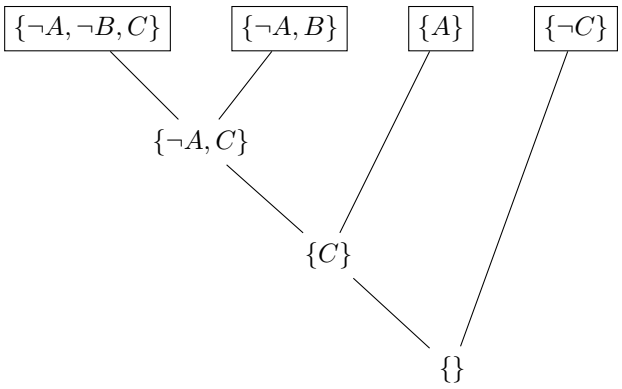
Resposta:

1. $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$
   Passagem para a forma clausal de $\neg((A \to (B \to C)) \to ((A \to B) \to (A \to C)))$:
   (a) Eliminação de $\to$: $\neg(\neg(\neg A \vee (\neg B \vee C)) \vee (\neg(\neg A \vee B) \vee (\neg A \vee C)))$
   (b) Redução do domínio de $\neg$: $\neg\neg(\neg A \vee (\neg B \vee C)) \wedge \neg(\neg(\neg A \vee B) \vee (\neg A \vee C))$
   $(\neg A \vee (\neg B \vee C)) \wedge (\neg\neg(\neg A \vee B) \wedge \neg(\neg A \vee C))$
   $(\neg A \vee (\neg B \vee C)) \wedge (\neg A \vee B) \wedge (\neg\neg A \wedge \neg C)$
   $(\neg A \vee (\neg B \vee C)) \wedge (\neg A \vee B) \wedge (A \wedge \neg C)$
   (c) Obtenção da forma conjuntiva normal: $(\neg A \vee \neg B \vee C) \wedge (\neg A \vee B) \wedge A \wedge \neg C$
   (d) Eliminação de $\wedge$: $\{\neg A \vee \neg B \vee C, \neg A \vee B, A, \neg C\}$
   (e) Eliminação de $\vee$: $\{\{\neg A, \neg B, C\}, \{\neg A, B\}, \{A\}, \{\neg C\}\}$

Aplicação do princípio da resolução:



ou



Nota: na segunda prova temos menos um passo porque aplicámos resolução para eliminar B e $\neg B$ a duas cláusulas que tinham $\neg A$ repetido.

2. $((A \to (A \to B)) \wedge A) \to B$
Passagem para a forma clausal de $\neg(((A \to (A \to B)) \wedge A) \to B)$:

   (a) Eliminação de $\to$: $\neg(\neg((\neg A \vee (\neg A \vee B)) \wedge A) \vee B)$

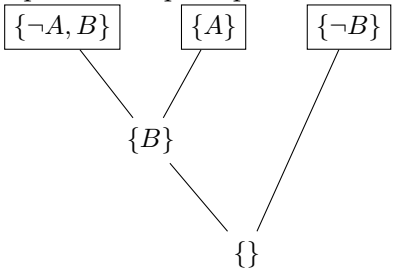
   (b) Redução do domínio de $\neg$: $\neg\neg((\neg A \vee (\neg A \vee B)) \wedge A) \wedge \neg B$
   $((\neg A \vee (\neg A \vee B)) \wedge A) \wedge \neg B$

   (c) Obtenção da forma conjuntiva normal: $(\neg A \vee \neg A \vee B) \wedge A \wedge \neg B$

   (d) Eliminação de $\wedge$: $\{\neg A \vee \neg A \vee B, A, \neg B\}$

   (e) Eliminação de $\vee$: $\{\{\neg A, B\}, \{A\}, \{\neg B\}\}$

   Aplicação do princípio da resolução:



3. $(A \to B) \to (\neg B \to \neg A)$
Passagem para a forma clausal de $\neg((A \to B) \to (\neg B \to \neg A))$:

(a) Eliminação de $\rightarrow$: $\neg(\neg(\neg A \vee B) \vee (\neg\neg B \vee \neg A))$

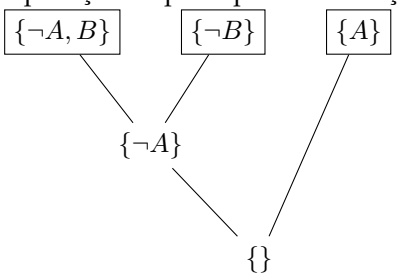
(b) Redução do domínio de $\neg$: $\neg(\neg(\neg A \vee B) \vee (B \vee \neg A))$
$\neg\neg(\neg A \vee B) \wedge \neg(B \vee \neg A)$
$(\neg A \vee B) \wedge (\neg B \wedge \neg\neg A)$
$(\neg A \vee B) \wedge (\neg B \wedge A)$

(c) Obtenção da forma conjuntiva normal: já está.

(d) Eliminação de $\wedge$: $\{\neg A \vee B, \neg B, A\}$

(e) Eliminação de $\vee$: $\{\{\neg A, B\}, \{\neg B\}, \{A\}\}$

Aplicação do princípio da resolução:



4. $(\neg B \rightarrow \neg A) \rightarrow (A \rightarrow B)$
Passagem para a forma clausal de $\neg((\neg B \rightarrow \neg A) \rightarrow (A \rightarrow B))$:

(a) Eliminação de $\rightarrow$: $\neg(\neg(\neg\neg B \vee \neg A) \vee (\neg A \vee B))$

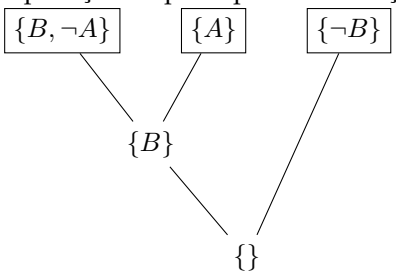
(b) Redução do domínio de $\neg$: $\neg(\neg(B \vee \neg A) \vee (\neg A \vee B))$
$\neg\neg(B \vee \neg A) \wedge \neg(\neg A \vee B)$
$(B \vee \neg A) \wedge (\neg\neg A \wedge \neg B)$
$(B \vee \neg A) \wedge (A \wedge \neg B)$

(c) Obtenção da forma conjuntiva normal: já está.

(d) Eliminação de $\wedge$: $\{B \vee \neg A, A, \neg B\}$

(e) Eliminação de $\vee$: $\{\{B, \neg A\}, \{A\}, \{\neg B\}\}$

Aplicação do princípio da resolução:



5. $(A \wedge B) \rightarrow \neg(A \rightarrow \neg B)$
Passagem para a forma clausal de $\neg((A \wedge B) \rightarrow \neg(A \rightarrow \neg B))$:

(a) Eliminação de $\rightarrow$: $\neg(\neg(A \wedge B) \vee \neg(\neg A \vee \neg B))$

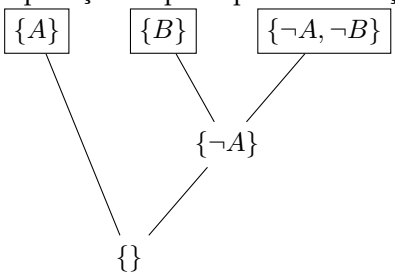
(b) Redução do domínio de $\neg$: $\neg\neg(A \wedge B) \wedge \neg\neg(\neg A \vee \neg B)$
$(A \wedge B) \wedge (\neg A \vee \neg B)$

(c) Obtenção da forma conjuntiva normal: já está.

(d) Eliminação de $\wedge$: $\{A, B, \neg A \vee \neg B\}$

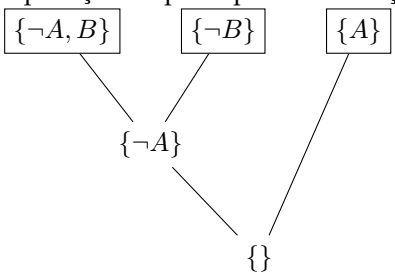
(e) Eliminação de $\vee$: $\{\{A\}, \{B\}, \{\neg A, \neg B\}\}$

Aplicação do princípio da resolução:



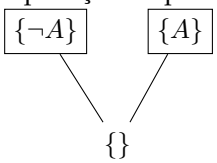
6. $((A \to B) \wedge \neg B) \to \neg A$
   Passagem para a forma clausal de $\neg(((A \to B) \wedge \neg B) \to \neg A)$:
   (a) Eliminação de $\to$: $\neg(\neg((\neg A \vee B) \wedge \neg B) \vee \neg A)$
   (b) Redução do domínio de $\neg$: $\neg\neg((\neg A \vee B) \wedge \neg B) \wedge \neg\neg A$
   $((\neg A \vee B) \wedge \neg B) \wedge A$
   (c) Obtenção da forma conjuntiva normal: já está.
   (d) Eliminação de $\wedge$: $\{\neg A \vee B, \neg B, A\}$
   (e) Eliminação de $\vee$: $\{\{\neg A, B\}, \{\neg B\}, \{A\}\}$

   Aplicação do princípio da resolução:



7. $(A \to \neg A) \to \neg A$
   Passagem para a forma clausal de $\neg((A \to \neg A) \to \neg A)$:
   (a) Eliminação de $\to$: $\neg(\neg(\neg A \vee \neg A) \vee \neg A)$
   (b) Redução do domínio de $\neg$: $\neg\neg(\neg A \vee \neg A) \wedge \neg\neg A$
   $\neg A \wedge A$
   (c) Obtenção da forma conjuntiva normal: já está.
   (d) Eliminação de $\wedge$: $\{\neg A, A\}$
   (e) Eliminação de $\vee$: $\{\{\neg A\}, \{A\}\}$

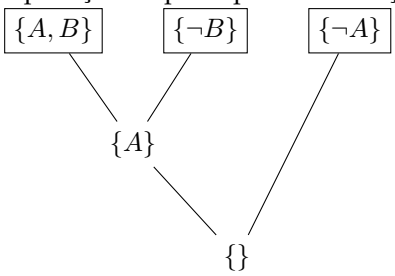
   Aplicação do princípio da resolução:



8. $(A \vee B) \to (B \vee A)$
   Passagem para a forma clausal de $\neg((A \vee B) \to (B \vee A))$:
   (a) Eliminação de $\to$: $\neg(\neg(A \vee B) \vee (B \vee A))$
   (b) Redução do domínio de $\neg$: $\neg\neg(A \vee B) \wedge \neg(B \vee A)$
   $(A \vee B) \wedge \neg B \wedge \neg A$
   (c) Obtenção da forma conjuntiva normal: já está.
   (d) Eliminação de $\wedge$: $\{A \vee B, \neg B, \neg A\}$

(e) Eliminação de $\vee$: $\{\{A, B\}, \{\neg B\}, \{\neg A\}\}$

Aplicação do princípio da resolução:



9. $((A \vee B) \vee C) \rightarrow (A \vee (B \vee C))$
Passagem para a forma clausal de $\neg(((A \vee B) \vee C) \rightarrow (A \vee (B \vee C)))$:

   (a) Eliminação de $\rightarrow$: $\neg(\neg((A \vee B) \vee C) \vee (A \vee (B \vee C)))$

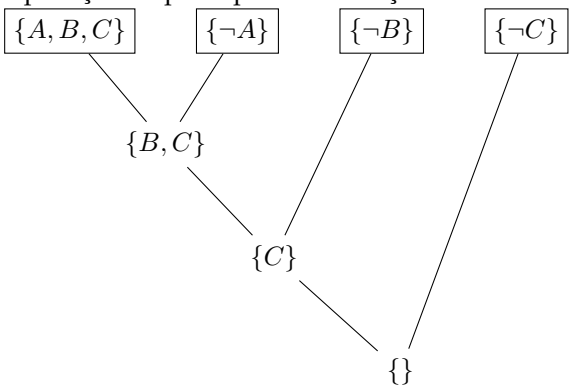
   (b) Redução do domínio de $\neg$: $\neg\neg((A \vee B) \vee C) \wedge \neg(A \vee (B \vee C))$
   $((A \vee B) \vee C) \wedge \neg(A \vee (B \vee C))$
   $((A \vee B) \vee C) \wedge \neg A \wedge \neg(B \vee C)$
   $((A \vee B) \vee C) \wedge \neg A \wedge \neg B \wedge \neg C$

   (c) Obtenção da forma conjuntiva normal: já está.

   (d) Eliminação de $\wedge$: $\{A \vee B \vee C, \neg A, \neg B, \neg C\}$

   (e) Eliminação de $\vee$: $\{\{A, B, C\}, \{\neg A\}, \{\neg B\}, \{\neg C\}\}$

Aplicação do princípio da resolução:



10. $(A \wedge (B \vee C)) \rightarrow ((A \wedge B) \vee (A \wedge C))$
Passagem para a forma clausal de $\neg((A \wedge (B \vee C)) \rightarrow ((A \wedge B) \vee (A \wedge C)))$:

   (a) Eliminação de $\rightarrow$: $\neg(\neg(A \wedge (B \vee C)) \vee ((A \wedge B) \vee (A \wedge C)))$

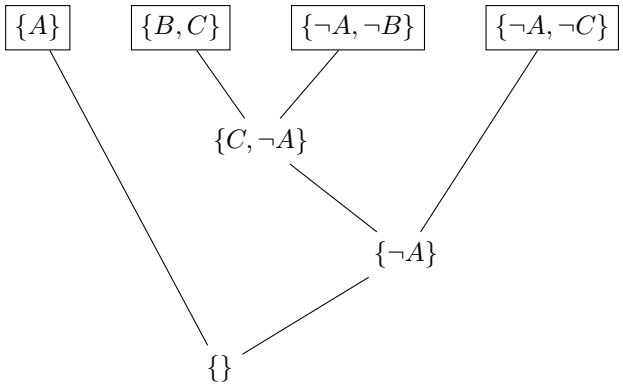
   (b) Redução do domínio de $\neg$: $\neg\neg(A \wedge (B \vee C)) \wedge \neg((A \wedge B) \vee (A \wedge C))$
   $(A \wedge (B \vee C)) \wedge \neg(A \wedge B) \wedge \neg(A \wedge C)$
   $A \wedge (B \vee C) \wedge (\neg A \vee \neg B) \wedge (\neg A \vee \neg C)$

   (c) Obtenção da forma conjuntiva normal: já está.

   (d) Eliminação de $\wedge$: $\{A, B \vee C, \neg A \vee \neg B, \neg A \vee \neg C\}$

   (e) Eliminação de $\vee$: $\{\{A\}, \{B, C\}, \{\neg A, \neg B\}, \{\neg A, \neg C\}\}$

Aplicação do princípio da resolução:



11. $((A \wedge B) \vee (A \wedge C)) \rightarrow (A \wedge (B \vee C))$
Passagem para a forma clausal de $\neg(((A \wedge B) \vee (A \wedge C)) \rightarrow (A \wedge (B \vee C)))$:

    (a) Eliminação de $\rightarrow$: $\neg(\neg((A \wedge B) \vee (A \wedge C)) \vee (A \wedge (B \vee C)))$

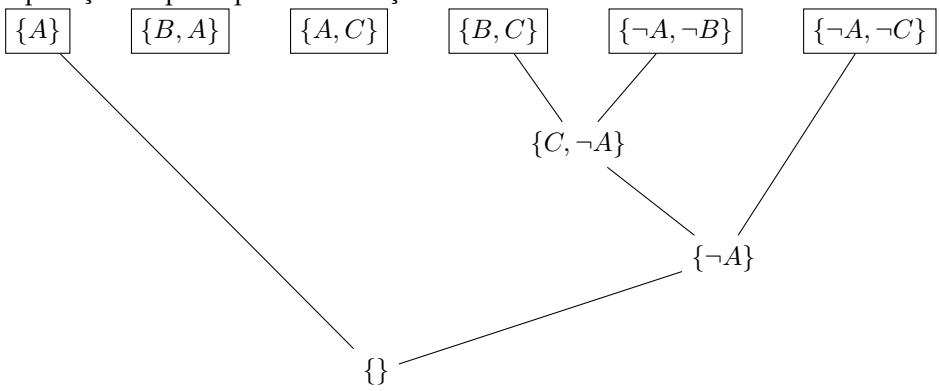
    (b) Redução do domínio de $\neg$: $\neg\neg((A \wedge B) \vee (A \wedge C)) \wedge \neg(A \wedge (B \vee C))$
    $((A \wedge B) \vee (A \wedge C)) \wedge (\neg A \vee \neg(B \vee C))$
    $((A \wedge B) \vee (A \wedge C)) \wedge (\neg A \vee (\neg B \wedge \neg C))$

    (c) Obtenção da forma conjuntiva normal:
    $(((A \wedge B) \vee A) \wedge ((A \wedge B) \vee C)) \wedge ((\neg A \vee \neg B) \wedge (\neg A \vee \neg C))$
    $((A \vee A) \wedge (B \vee A)) \wedge ((A \vee C) \wedge (B \vee C)) \wedge ((\neg A \vee \neg B) \wedge (\neg A \vee \neg C))$

    (d) Eliminação de $\wedge$: $\{A, B \vee A, A \vee C, B \vee C, \neg A \vee \neg B, \neg A \vee \neg C\}$

    (e) Eliminação de $\vee$: $\{\{A\}, \{B, A\}, \{A, C\}, \{B, C\}, \{\neg A, \neg B\}, \{\neg A, \neg C\}\}$

    Aplicação do princípio da resolução:



    Nota: aqui também não foi necessário usar todas as premissas para chegar a uma contradição. Neste caso, as duas premissas que não foram usadas até poderiam ter sido eliminadas usando a estratégia de eliminação de cláusulas não mínimas, uma vez que ambas contêm a premissa $\{A\}$.

12. $((A \wedge B) \vee A) \rightarrow A$
Passagem para a forma clausal de $\neg(((A \wedge B) \vee A) \rightarrow A)$:

    (a) Eliminação de $\rightarrow$: $\neg(\neg((A \wedge B) \vee A) \vee A)$

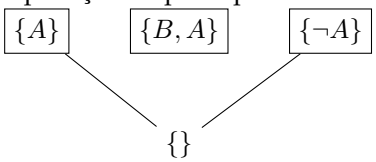
    (b) Redução do domínio de $\neg$: $\neg\neg((A \wedge B) \vee A) \wedge \neg A$
    $((A \wedge B) \vee A) \wedge \neg A$

    (c) Obtenção da forma conjuntiva normal: $(A \vee A) \wedge (B \vee A) \wedge \neg A$

    (d) Eliminação de $\wedge$: $\{A, B \vee A, \neg A\}$

    (e) Eliminação de $\vee$: $\{\{A\}, \{B, A\}, \{\neg A\}\}$

Aplicação do princípio da resolução:

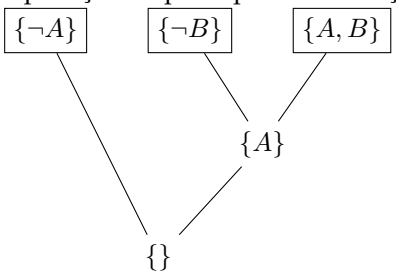


Exercício 4.4
Demonstre os teoremas do exercício 2.3, que correspondem a aplicações das leis de De Morgan, usando resolução.

Resposta:

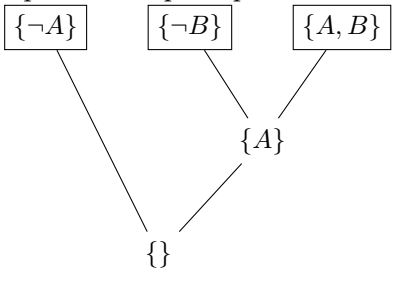
1. $\neg(A \vee B) \rightarrow (\neg A \wedge \neg B)$
   Passagem para a forma clausal de $\neg(\neg(A \vee B) \rightarrow (\neg A \wedge \neg B))$:
   (a) Eliminação de $\rightarrow$: $\neg(\neg\neg(A \vee B) \vee (\neg A \wedge \neg B))$
   (b) Redução do domínio de $\neg$: $\neg(A \vee B) \wedge \neg(\neg A \wedge \neg B)$
   $(\neg A \wedge \neg B) \wedge (\neg\neg A \vee \neg\neg B)$
   $(\neg A \wedge \neg B) \wedge (A \vee B)$
   (c) Obtenção da forma conjuntiva normal: já está.
   (d) Eliminação de $\wedge$: $\{\neg A, \neg B, A \vee B\}$
   (e) Eliminação de $\vee$: $\{\{\neg A\}, \{\neg B\}, \{A, B\}\}$

   Aplicação do princípio da resolução:



2. $(\neg A \wedge \neg B) \rightarrow \neg(A \vee B)$
   Passagem para a forma clausal de $\neg((\neg A \wedge \neg B) \rightarrow \neg(A \vee B))$:
   (a) Eliminação de $\rightarrow$: $\neg(\neg(\neg A \wedge \neg B) \vee \neg(A \vee B))$
   (b) Redução do domínio de $\neg$: $\neg\neg(\neg A \wedge \neg B) \wedge \neg\neg(A \vee B)$
   $(\neg A \wedge \neg B) \wedge (A \vee B)$
   (c) Obtenção da forma conjuntiva normal: já está.
   (d) Eliminação de $\wedge$: $\{\neg A, \neg B, A \vee B\}$
   (e) Eliminação de $\vee$: $\{\{\neg A\}, \{\neg B\}, \{A, B\}\}$

   Aplicação do princípio da resolução:



3. $\neg(A \wedge B) \rightarrow (\neg A \vee \neg B)$
   Passagem para a forma clausal de $\neg(\neg(A \wedge B) \rightarrow (\neg A \vee \neg B))$:

   (a) Eliminação de $\rightarrow$: $\neg(\neg\neg(A \wedge B) \vee (\neg A \vee \neg B))$

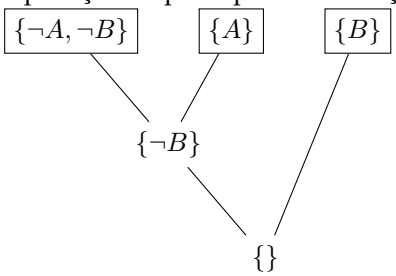
   (b) Redução do domínio de $\neg$: $\neg(A \wedge B) \wedge \neg(\neg A \vee \neg B)$
   $(\neg A \vee \neg B) \wedge (A \wedge B)$

   (c) Obtenção da forma conjuntiva normal: já está.

   (d) Eliminação de $\wedge$: $\{\neg A \vee \neg B, A, B\}$

   (e) Eliminação de $\vee$: $\{\{\neg A, \neg B\}, \{A\}, \{B\}\}$

   Aplicação do princípio da resolução:



4. $(\neg A \vee \neg B) \rightarrow \neg(A \wedge B)$
   Passagem para a forma clausal de $\neg((\neg A \vee \neg B) \rightarrow \neg(A \wedge B))$:

   (a) Eliminação de $\rightarrow$: $\neg(\neg(\neg A \vee \neg B) \vee \neg(A \wedge B))$

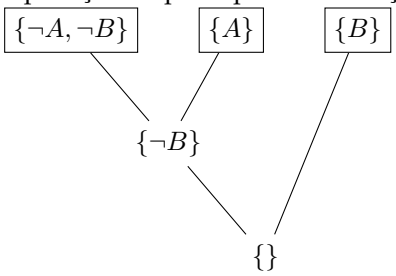
   (b) Redução do domínio de $\neg$: $\neg\neg(\neg A \vee \neg B) \wedge \neg\neg(A \wedge B)$
   $(\neg A \vee \neg B) \wedge (A \wedge B)$

   (c) Obtenção da forma conjuntiva normal: já está.

   (d) Eliminação de $\wedge$: $\{\neg A \vee \neg B, A, B\}$

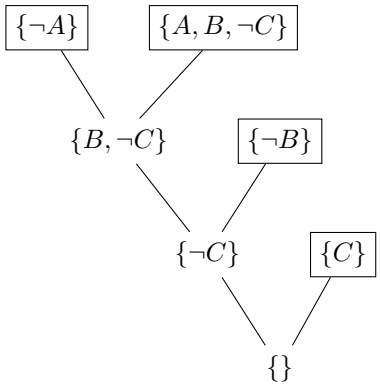
   (e) Eliminação de $\vee$: $\{\{\neg A, \neg B\}, \{A\}, \{B\}\}$

   Aplicação do princípio da resolução:



Exercício 4.5
Usando uma estratégia de resolução linear, apresente uma prova por refutação para A a partir do seguinte conjunto de cláusulas: $\{\{A, B, \neg C\}, \{\neg D, A\}, \{\neg B\}, \{C\}\}$.

Resposta:



Nota: não foi necessário usar a cláusula $\{\neg D, A\}$. Aliás, se a tivéssemos usado no primeiro passo não teríamos conseguido fazer o resto da prova usando resolução linear, pois não conseguiríamos eliminar $\neg D$. A cláusula $\{\neg D, A\}$ poderia até ser eliminada usando a estratégia de eliminação de literais puros.

5 Lógica Proposicional — BDDs

Sumário:

- Provas usando o sistema semântico da lógica proposicional
- Árvores de decisão
- BDDs

Resumo:

As *árvores de decisão* correspondem à versão gráfica das tabelas de verdade. Uma árvore de decisão para uma fbf é uma árvore binária em que os nós contêm os símbolos de proposição existentes na fbf e as folhas da árvore são um dos valores $\boxed{\text{V}}$ e $\boxed{\text{F}}$.

Os *BDDs* (Binary Decision Diagrams) são grafos acíclicos dirigidos e rotulados que representam de forma compacta a mesma informação que as árvores de decisão. As árvores de decisão em si também são BDDs, antes de serem simplificados.

Transformações aplicáveis a BDDs:

R1 - Remoção de folhas duplicadas: Todos os nós com rótulos $\boxed{\text{V}}$ e $\boxed{\text{F}}$ são unificados, e os arcos são redireccionados para os novos nós.

R2 - Remoção de testes redundantes: Se ambos os arcos que saem dum nó vão para o mesmo nó, podemos eliminar o primeiro, redireccionando os arcos que apontam para ele.

R3 - Remoção de nós redundantes: Se dois nós distintos são os nós iniciais de dois sub-BDDs estruturalmente semelhantes, então podemos eliminar um deles, dirigindo os nós relevantes do outro.

Composição de BDDs:

1. O BDD para a fbf $\neg\alpha$ pode ser criado a partir do BDD_α, substituindo o nó com o rótulo $\boxed{\text{V}}$ por $\boxed{\text{F}}$ e vice-versa.
2. O BDD para a fbf $\alpha \wedge \beta$ pode ser criado a partir do BDD_α, substituindo o nó com o rótulo $\boxed{\text{V}}$ pelo BDD_β e aplicando ao BDD resultante as reduções R1 a R3.
3. O BDD para a fbf $\alpha \vee \beta$ pode ser criado a partir do BDD_α, substituindo o nó com o rótulo $\boxed{\text{F}}$ pelo BDD_β e aplicando ao BDD resultante as reduções R1 a R3.

Da composição de BDDs podem resultar BDDs em que existe mais do que um nó para o mesmo símbolo de predicado. Isto não é problemático, mas é necessário considerar apenas os caminhos consistentes. Para resolver este problema introduzem-se os OBDDs.

Exercício 5.1
Considere a seguinte tabela de verdade:

P	Q	$f(P,Q)$
V	V	V
V	F	F
F	V	F
F	F	F

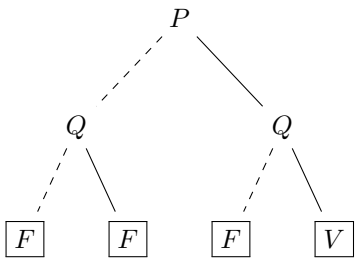
1. Represente a árvore de decisão correspondente.

2. Mostre o BDD reduzido correspondente à árvore anterior, apresentando e justificando todos os passos.

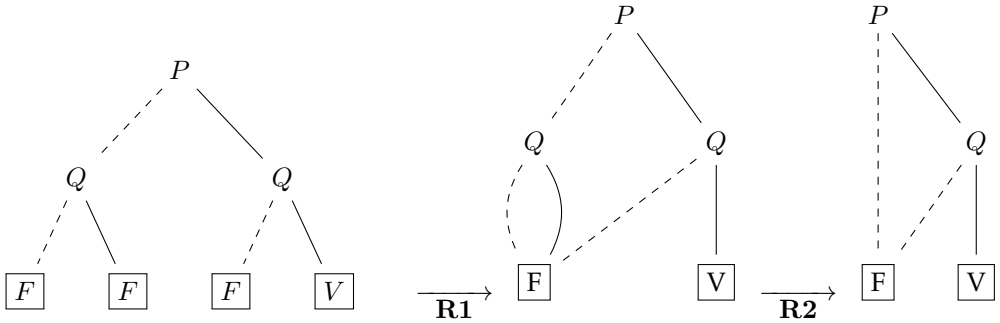
3. Com base no BDD reduzido, diga quais são os modelos de $f(P,Q)$ e compare-os com os obtidos pela observação da tabela de verdade.

Resposta:

1.



2.



3. Seguindo o caminho desde a raíz do BDD até ao nó $\boxed{\text{V}}$, observa-se que, para $f(P,Q)$ ser verdadeira, P e Q têm ambos que ser verdadeiros, o que é o mesmo que seria concluido observando a tabela de verdade (como seria de esperar!).

Exercício 5.2
Considere a seguinte tabela de verdade:

P	Q	R	$f(P,Q,R)$
V	V	V	V
V	V	F	F
V	F	V	F
V	F	F	F
F	V	V	V
F	V	F	F
F	F	V	V
F	F	F	F

1. Represente a árvore de decisão correspondente.

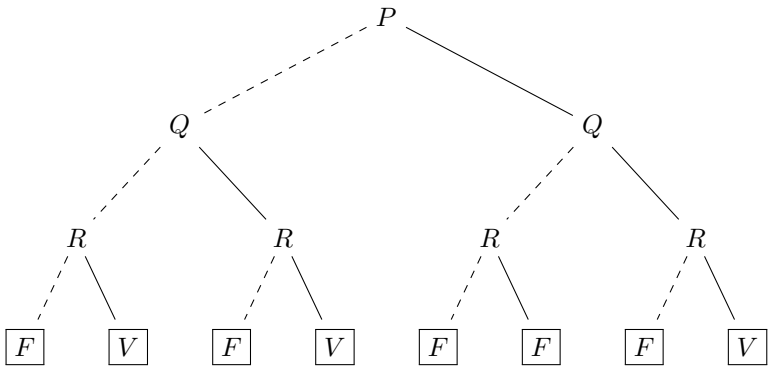
2. Mostre o BDD reduzido correspondente à árvore anterior, apresentando e justificando todos os passos.

3. O BDD teria a mesma forma se tivesse escolhido outra ordenação para os nós? Justifique, mostrando o novo BDD reduzido.

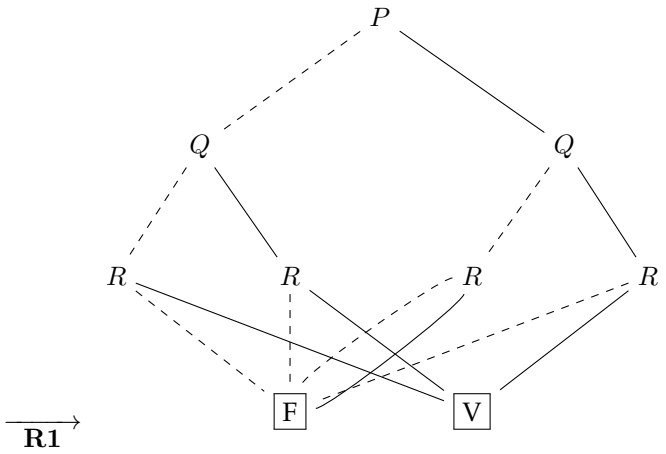
4. Com base nos BDDs reduzidos, diga quais são os modelos de $f(P,Q,R)$ e compare-os com os obtidos pela observação da tabela de verdade.

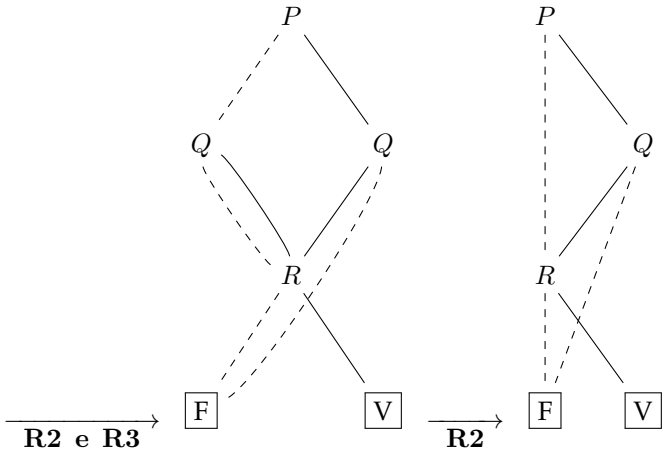
Resposta:

1.

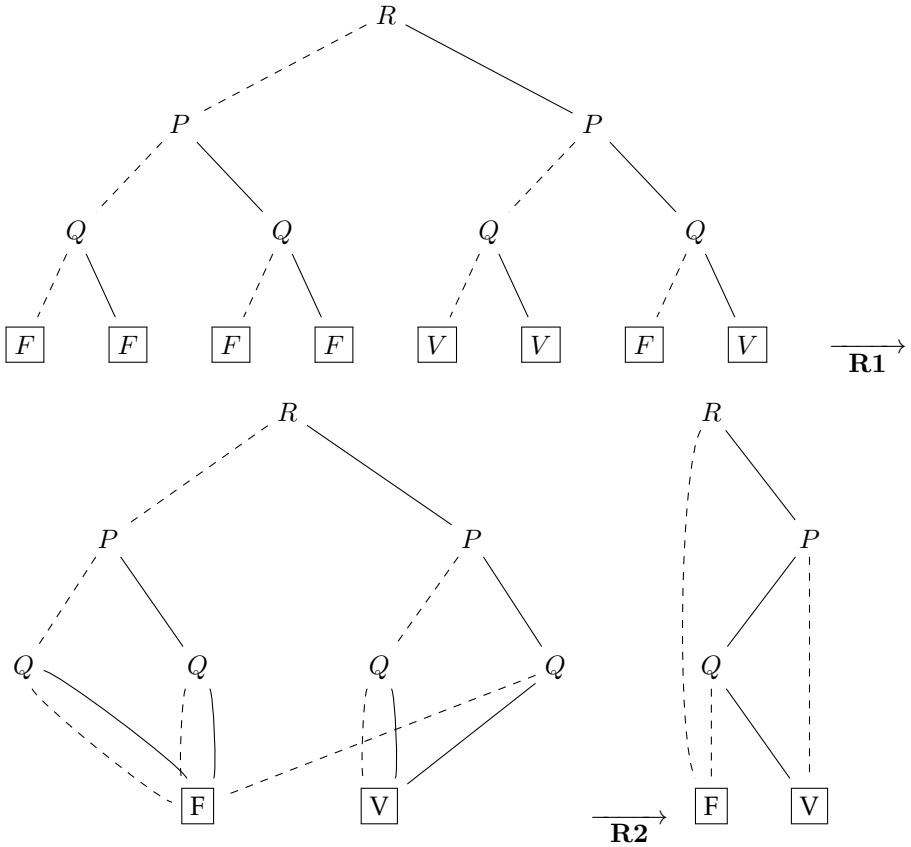


2.





3. Para saber se a forma do BDD reduzido seria a mesma usando outra ordenação, vamos voltar a fazer a árvore de decisão e o BDD reduzido usando outra ordenação.



Nota: para a mesma tabela de verdade, temos 2 BDDs reduzidos diferentes!

4. Seguindo o caminho desde a raíz de cada BDD até ao nó $\boxed{\text{V}}$, observa-se que, para $f(P, Q, R)$ ser verdadeira, existem duas possibilidades: P, Q, R são todos verdadeiros; ou P é falso e R é verdadeiro (neste caso, não interessa o valor de Q). Como seria de esperar, a conclusão é a mesma, observando o primeiro BDD, o segundo BDD ou a tabela de verdade.

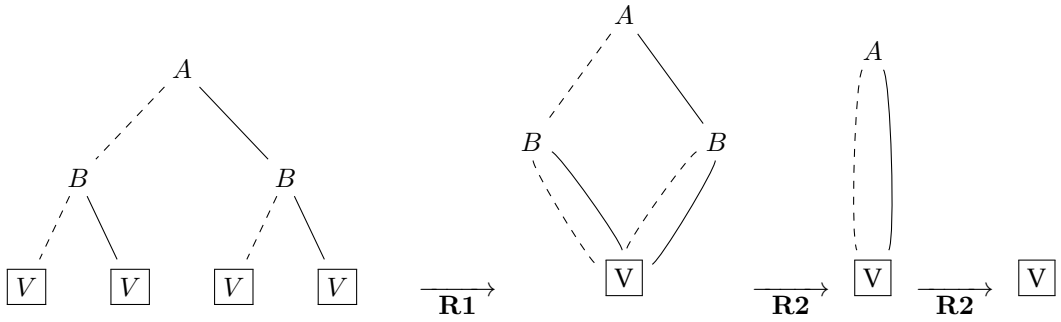
Exercício 5.3
Usando BDDs, prove que as seguintes fórmulas, correspondentes às leis de De Morgan, são tautologias.

$$\neg(A \vee B) \leftrightarrow (\neg A \wedge \neg B)$$

$$\neg(A \wedge B) \leftrightarrow (\neg A \vee \neg B)$$

Resposta:

Mostrar que, para cada uma das fórmulas, o BDD reduzido é o BDD de uma tautologia, ou seja tem apenas a folha $\boxed{\text{V}}$. Como todas as interpretações das duas fbfs têm o valor verdadeiro e as fbfs usam os mesmos símbolos proposicionais, ambas terão a mesma árvore de decisão.

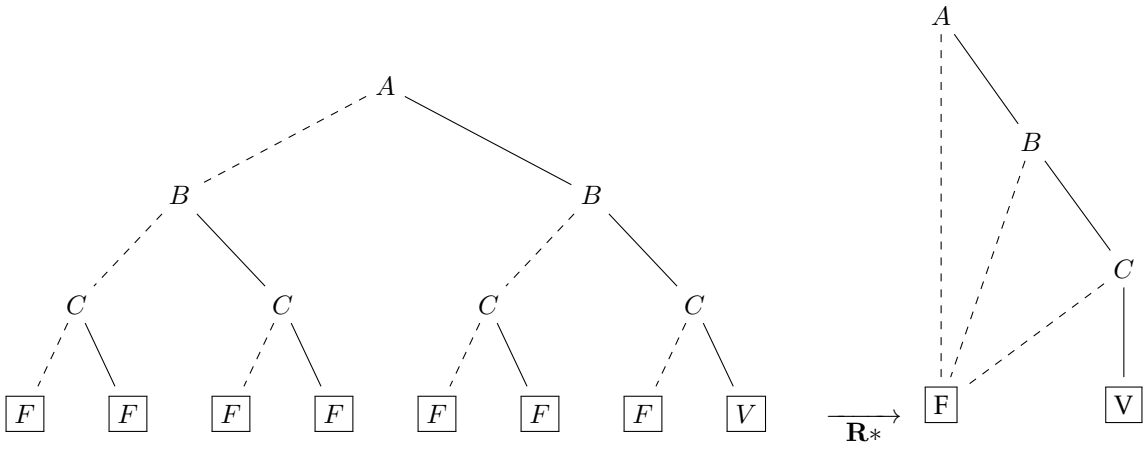


Exercício 5.4
Considere que $\alpha = A \wedge B \wedge C$ e $\beta = B \vee C$. Usando as regras de composição de BDDs, determine os BDDs reduzidos para:

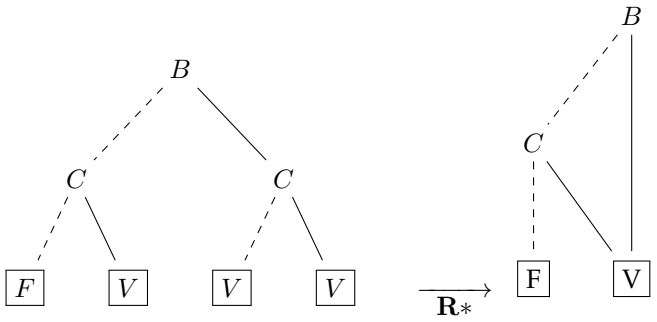
1. α
2. β
3. $\alpha \wedge \beta$
4. $\alpha \vee \neg\beta$

Resposta:

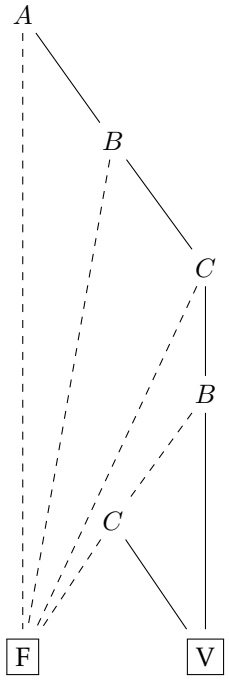
1. $\alpha = A \wedge B \wedge C$



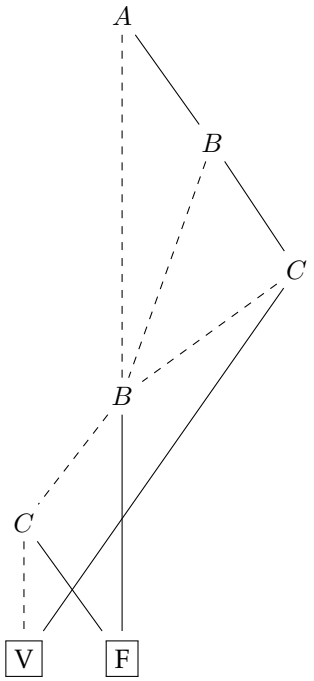
2. $\beta = B \vee C$



3. $\alpha \wedge \beta = (A \wedge B \wedge C) \wedge (B \vee C)$



4. $\alpha \vee \neg\beta = (A \wedge B \wedge C) \vee \neg(B \vee C)$



Nota: o problema com as duas últimas alíneas, que correspondem à composição simples de BDDs, é que temos caminhos por onde é necessário testar mais do que uma vez o valor de verdade de um símbolo proposicional, e pior ainda é que no mesmo caminho consideramos que um símbolo proposicional é verdadeiro e mais tarde que é falso (ou vice-versa), o que não faz sentido. Os OBBDs vêm resolver estes problemas. Convém também notar que estes problemas não existiriam se as fbfs α e β não tivessem símbolos proposicionais em comum.

6 Lógica Proposicional — OBDDs

Sumário:

- Provas usando o sistema semântico da lógica proposicional
- OBDDs

Resumo:

Da composição de BDDs podem resultar BDDs em que existe mais do que um nó para o mesmo símbolo proposicional no mesmo caminho, no caso em que as fbfs a compor partilhem símbolos proposicionais. Isto não é problemático, apesar de ser ineficiente, mas é necessário considerar apenas os caminhos consistentes. Para além disso, é difícil verificar a equivalência de BDDs com ordenações diferentes para os símbolos de predicado usados. Para resolver estes problemas introduzem-se os OBDDs.

Um OBBD é um BDD que que satisfaz alguma relação de ordem total para os símbolos proposicionais que contém. Usando OBBDs, cada fbf tem uma representação única, ou seja, tem uma forma canónica, que é sempre a mesma, independentemente da ordem pela qual as reduções forem aplicadas.

Dados $OBDD1$ e $OBDD2$ e um conjunto de símbolos de proposição $\{P_1, \ldots, P_n\}$ contendo os símbolos de proposição de $OBDD1$ e $OBDD2$, diz-se que $OBDD1$ e $OBDD2$ são *compatíveis* sse existe uma ordem $\prec$ aplicada ao conjunto de símbolos de proposição tal que ambos os OBDDs satisfazem essa ordem.

A ordenação escolhida para os predicados pode influenciar (e muito!) o tamanho do OBBD correspondente.

Dada uma fbf α e uma relação de ordem total para os símbolos de proposição de α, o OBDD reduzido correspondente a α é único. Consequentemente, é possível saber se dois OBDDs correspondem à mesma fbf verificando se estes são estruturalmente semelhantes.

Teste para tautologia/validade — uma fbf α é tautológica/válida (isto é, tem sempre o valor verdadeiro) sse o seu OBBD reduzido for $\boxed{\text{V}}$.

Teste para inconsistência — uma fbf α é contraditória sse o seu OBBD reduzido for $\boxed{\text{F}}$.

Teste para satisfazibilidade — uma fbf α é satisfazível sse o seu OBBD reduzido não for $\boxed{\text{F}}$, ou seja, se tiver pelo menos um caminho para a folha $\boxed{\text{V}}$.

Teste de equivalência semântica — duas fbfs são equivalentes sse os seus OBBDs reduzidos são estruturalmente semelhantes.

Ausência de variáveis redundantes — se o valor de uma fbf α não depende de x_i, então qualquer OBBD reduzido para α não vai ter nenhum nó para x_i.

Algoritmo *reduz*

O algoritmo *reduz* percorre o grafo correspondente ao OBDD por níveis, começando nos nós terminais (folhas). Ao percorrer o grafo, atribui um identificador a cada OBDD de tal

modo que dois OBDDs representam a mesma fbf sse os seus identificadores são iguais, ou seja, se $id(o_1) = id(o_2)$.

1. Todas as folhas $\boxed{\text{F}}$ recebem o identificador 0 e todas as folhas $\boxed{\text{V}}$ recebem o identificador 1.
2. A atribuição de identificadores aos OBDDs do nível i é feita assumindo que o algoritmo atribuiu identificadores a todos os OBDDs do nível $i + 1$. Dado um OBDD, o, ao nível i:
   (a) Se $id(neg(o)) = id(pos(o))$, então $mudaId(o, id(neg(o)))$ pois a raiz deste OBDD efectua um teste redundante e pode ser removida pela transformação R2.
   (b) Se, entre os OBDDs que já têm identificador existe um o_{Id} tal que $(raiz(o) = raiz(o_{Id}))$, $id(neg(o)) = id(neg(o_{Id}))$ e $id(pos(o)) = id(pos(o_{Id}))$ então $mudaId(o, id(o_{Id}))$, pois este OBDD corresponde a um OBDD redundante e pode ser removido pela transformação R3.
   (c) Caso contrário, o OBDD o recebe um novo identificador.
3. Após a atribuição de identificadores a todos os sub-OBDDs, o OBDD resultante é compactado.

Algoritmo *aplica*

O algoritmo *aplica* recebe um operador lógico, *op*, e dois OBDDs reduzidos e compatíveis o_α e o_β, correspondentes às fbfs α e β, e devolve o OBDD reduzido correspondente à fbf $(\alpha\ op\ \beta)$. Nota: Para o *aplica* poder ser definido apenas para operadores binários, considera-se $(\neg\alpha) \leftrightarrow (\alpha \oplus V)$, em que $\oplus$ corresponde ao *ou exclusivo*.

A intuição subjacente a este algoritmo é a seguinte:

1. Se ambos os ODBBs corresponderem a folhas, aplica-se a operação op aos correspondentes valores lógicos.
2. Caso contrário, escolhe-se o símbolo de proposição com maior prioridade existente em o_α e o_β e divide-se o problema em dois subproblemas mais simples, num dos quais o predicado é verdadeiro e no outro o predicado é falso.
   (a) Se o símbolo de proposição é a raiz de ambos os OBDDs, o OBDD resultante terá esse símbolo como raiz, o seu OBDD negativo resulta de aplicar recursivamente o algoritmo aos correspondentes OBDDs negativos e analogamente para o seu OBDD positivo.
   (b) Caso contrário, o OBDD resultante terá esse símbolo de proposição como raiz, o seu OBDD negativo resulta de aplicar recursivamente o algoritmo ao OBDD negativo do OBDD que contém o símbolo e ao outro OBDD (que não contém o símbolo), e analogamente para o OBDD positivo.

No fim, reduzir o OBBD resultante dos passos anteriores.

Notação para $aplica(op, o_\alpha, o_\beta) = o_{(\alpha\ op\ \beta)}$

$$\boxed{o_\alpha\ op\ o_\beta}$$
$$=$$
$$o_{(\alpha\ op\ \beta)}$$

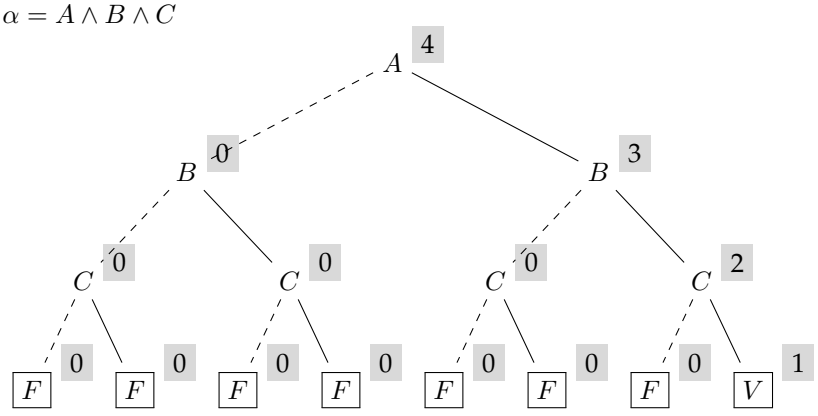
Exercício 6.1
Considere as seguintes fbfs:

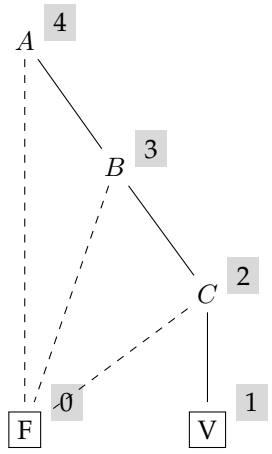
1. $\alpha = A \wedge B \wedge C$
2. $\beta = B \vee C$
3. $\gamma = (A \wedge B) \vee D$

Partindo das suas árvores de decisão binárias, mostre os seus OBBDs reduzidos através da aplicação do algoritmo *reduz*. Deve usar a ordenação $[A, B, C, D]$ para os predicados.

Resposta:

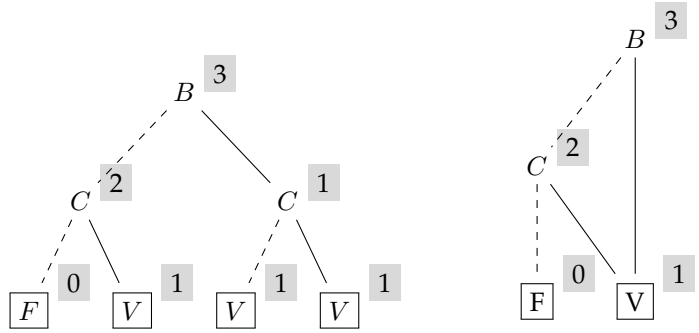
1. $\alpha = A \wedge B \wedge C$



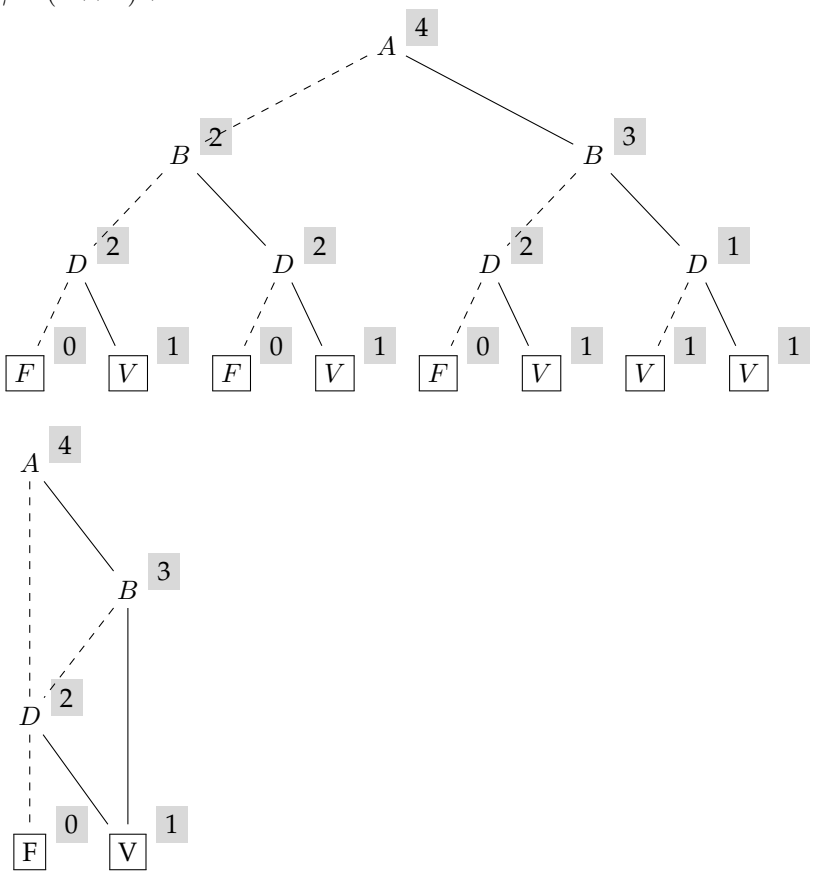


Nota: obtém-se um OBDD igual ao BDD reduzido da aula passada, mas com apenas uma passagem pelo OBDD original. O mesmo vai acontecer na próxima alínea.

2. $\beta = B \vee C$



3. $\gamma = (A \wedge B) \vee D$



Exercício 6.2
Considerando os OBBDs reduzidos do exercício anterior, combine-os usando o algoritmo *aplica* (se isso for possível) para obter os OBBDs reduzidos para as seguintes fbfs.

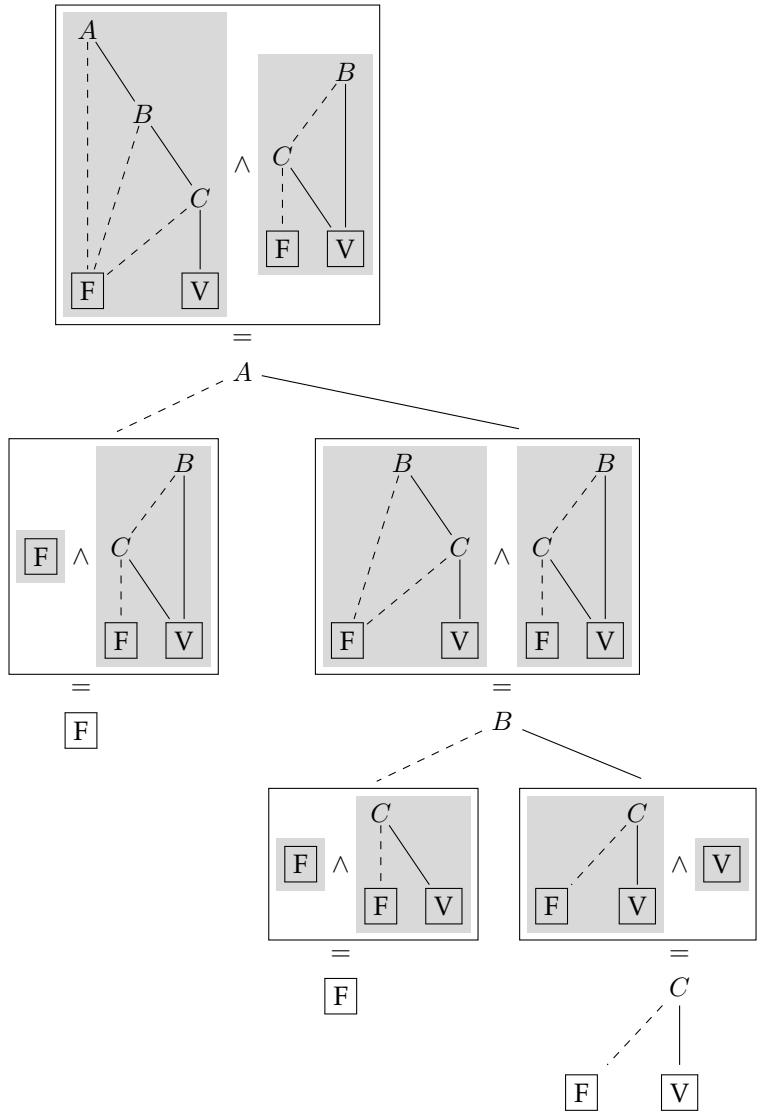
1. $\alpha \wedge \beta$
2. $\alpha \vee \neg\beta$
3. $\beta \vee \gamma$

Resposta:

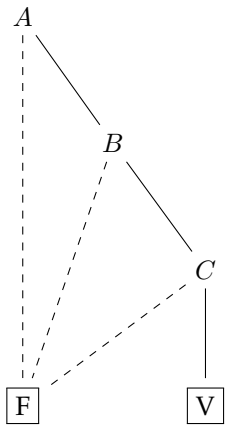
É possível combinar os OBBDs porque eles são compatíveis, isto é, têm ordenações que permitem a sua combinação.

1. $\alpha \wedge \beta = (A \wedge B \wedge C) \wedge (B \vee C)$

   Para determinar o OBDD de $\alpha \wedge \beta$, vamos calcular $aplica(\wedge, o_\alpha, o_\beta)$.



Reconstruindo o resultado através das chamadas recursivas ao *aplica*, obtém-se o OBBD:

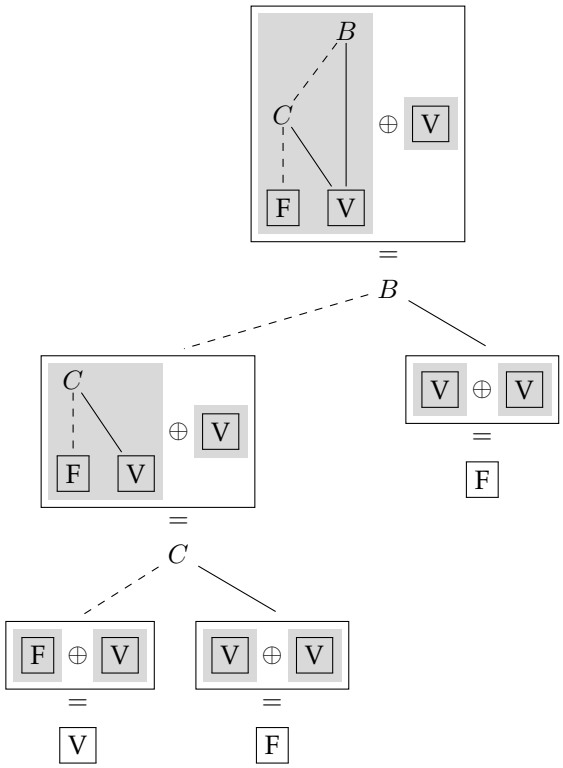


Este OBDD (que por acaso é igual ao OBBD de α), é mais compacto do que o resultante da combinação da aula passada, não tem nós repetidos e consequentemente não tem caminhos inconsistentes.

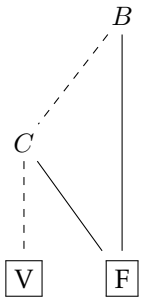
2. $\alpha \vee \neg\beta = (A \wedge B \wedge C) \vee \neg(B \vee C)$

Para determinar o OBBD de $\alpha \vee \neg\beta$, é necessário primeiro determinar o OBBD de $\neg\beta$, que é equivalente a $\beta \oplus V$. Assim, para determinar o OBDD desta fórmula, vamos calcular $aplica(\vee, o_\alpha, aplica(\oplus, o_\beta, \boxed{\text{V}}))$.

Comecemos por calcular $aplica(\oplus, o_\beta, \boxed{\text{V}})$.

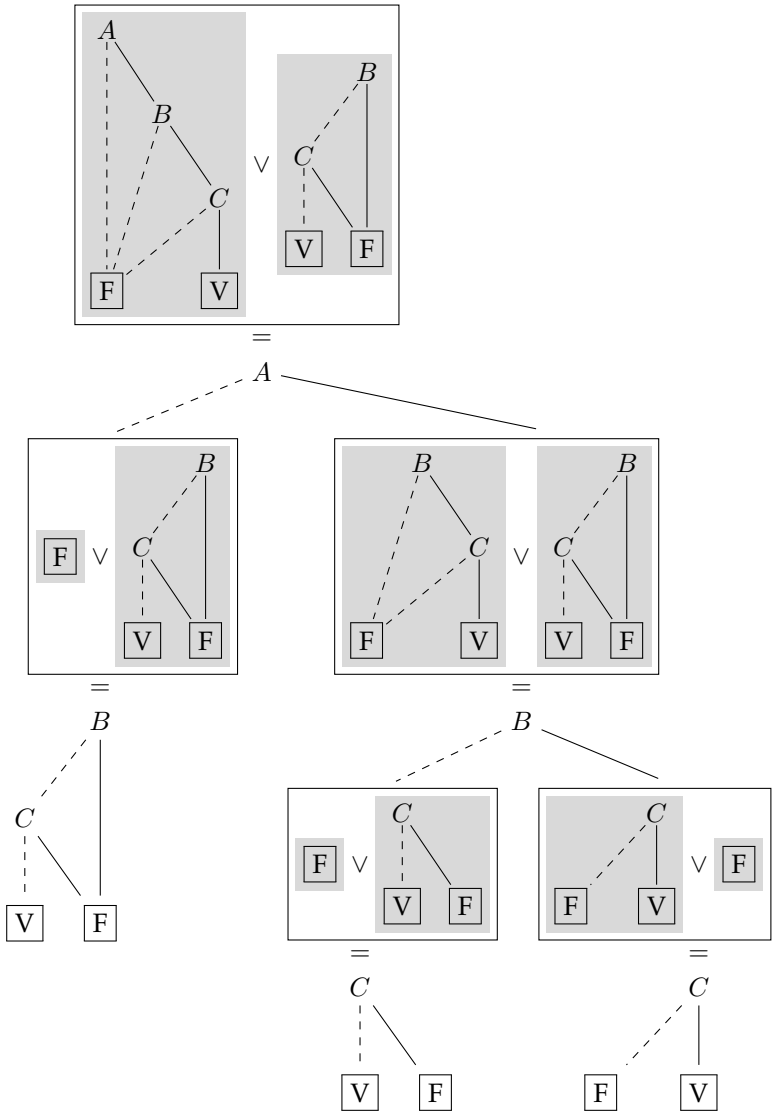


Reconstruindo o resultado através das chamadas recursivas ao *aplica*, obtém-se o OBBD:

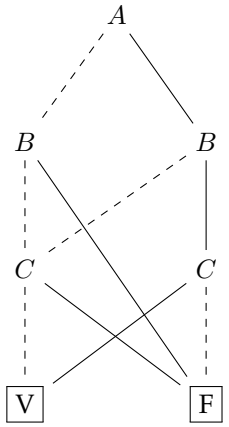


Tal como seria de esperar, este OBDD corresponde ao BDD de β trocando $\boxed{\text{V}}$ por $\boxed{\text{F}}$ e vice-versa.

Para determinar o OBBD de $\alpha \vee \neg\beta$ podemos agora calcular $aplica(\vee, o_\alpha, o_{\neg\beta})$.

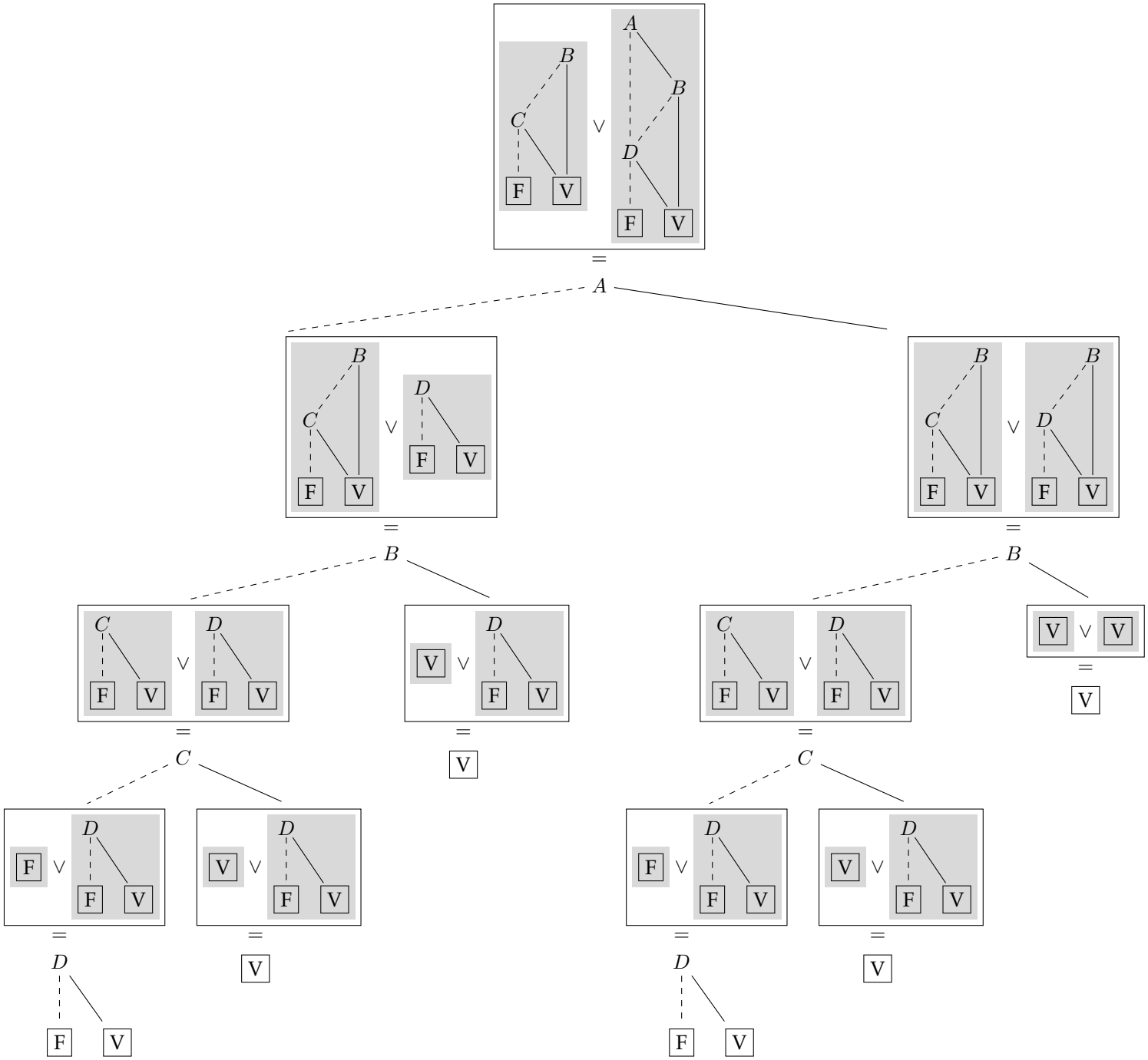


Reconstruindo o resultado através das chamadas recursivas ao *aplica*, obtém-se o OBBD:

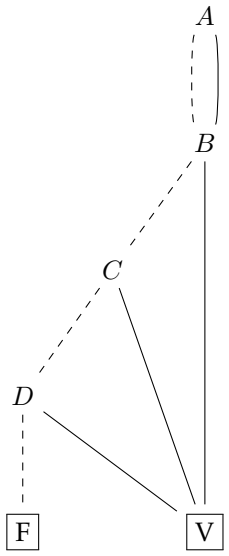


3. $\beta \vee \gamma = (B \vee C) \vee (A \wedge B) \vee D$

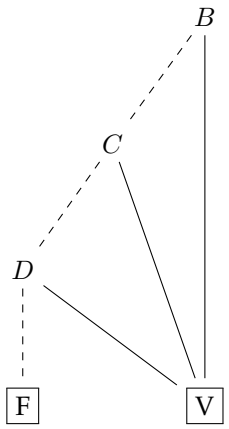
Para determinar o OBDD de $\beta \vee \gamma$, vamos calcular $aplica(\vee, o_\beta, o_\gamma)$.



Reconstruindo o resultado através das chamadas recursivas ao *aplica*, obtém-se o OBBD:



Que ainda pode ser reduzido, uma vez que ambos os arcos do nó A apontam para o mesmo nó B.



7 Lógica Proposicional — SAT

Sumário:

- Sistema semântico da lógica proposicional
- Algoritmos de propagação de marcas e de teste de nós
- Algoritmo DP

Resumo:

Em SAT, pretende-se determinar se uma fbf é satisfazível, e em caso afirmativo determinar uma interpretação que a satisfaça (a testemunha). Os BDDs já encontram todas as interpretações que satisfazem uma fbf, o interesse de SAT é que é mais eficiente.

Para provar que uma fórmula é uma tautologia, negamos a fórmula inicial e tentamos chegar a uma contradição. Para provar que um argumento é válido, criamos uma fórmula com a conjunção das premissas e a negação da conclusão e tentamos chegar a uma contradição.

Algoritmo de propagação de marcas

Este algoritmo é muito eficiente (apresenta uma ordem de crescimento linear em função do número de símbolos de proposição existentes na fbf), mas não é completo, porque não funciona para fórmulas da forma $\neg(\alpha \wedge \beta)$.

Se as fbfs contiverem apenas conjunções e negações, podemos usar um algoritmo de propagação de marcas que propaga as restrições através do DAG que representa a fbf.

1. Transformar a fórmula para só conter conjunções e negações:
   traduzir $\alpha \vee \beta$ para $\neg(\neg\alpha \wedge \neg\beta)$,
   traduzir $\alpha \rightarrow \beta$ para $\neg(\alpha \wedge \neg\beta)$ e
   eliminar duplas negações, traduzindo $\neg\neg\alpha$ para α.
2. Construir o DAG, partilhando as folhas e nós repetidos.
3. Propagar as marcas, usando o algoritmo de propagação de marcas e, se necessário, o algoritmo de teste de nós. Os inteiros que aparecem associados às marcas indicam a ordem pela qual as marcas foram atribuídas aos respectivos nós.
4. Verificar as marcas: se conseguirmos marcar o DAG sem contradições, propagar as marcas das folhas para cima e verificar se realmente as marcas que colocámos estão correctas. Se tudo estiver consistente, encontrámos a *testemunha*, que é uma interpretação que satisfaz a fórmula inicial, e que corresponde às marcas que estão nas folhas do DAG. Se encontrámos marcas contraditórias, a fórmula inicial não é satisfazível (ou é contraditória).

Algoritmo de teste de nós

Quando o algoritmo de propagação de marcas não consegue marcar todos os nós de um grafo, deve ser aplicado o algoritmo de teste de nós, que tem uma ordem de crescimento

cúbica em relação ao número de símbolos proposicionais da fbf. Este algoritmo escolhe um dos nós por marcar e testa esse nó, marcando-o temporariamente com um valor lógico, efectuando a propagação desta marca temporária e determinando eventualmente outras marcas temporárias. Se este teste não permitir resolver o problema, o mesmo nó é testado com o valor lógico contrário. Após o teste de um nó, várias situações podem ocorrer:

1. Se todos os nós ficaram marcados com marcas consistentes, o algoritmo termina, pois foi encontrada uma testemunha;
2. Se foi encontrada uma contradição, então o nó é marcado permanentemente com a marca contrária à que foi usada no teste, e é utilizado o algoritmo de propagação de marcas;
3. Se nenhuma das situações anteriores se verifica, são comparadas as marcas obtidas nos dois testes do nó; as marcas (temporárias) comuns aos dois testes são passadas a permanentes, e propagadas pelo algoritmo de propagação de marcas.

Algoritmo DP

Este algoritmo é menos eficiente do que o anterior (tem ordem de crescimento exponencial), mas é correcto e completo.

O algoritmo DP serve para determinar se um determinado conjunto de cláusulas é satisfazível e tal como a resolução utiliza regras que transformam conjuntos de cláusulas em conjuntos de cláusulas. Relativamente à resolução, tem a vantagem de permitir determinar uma interpretação que satisfaz o conjunto de cláusulas.

Uma maneira de implementar o algoritmo DP é usar a *eliminação de baldes*:

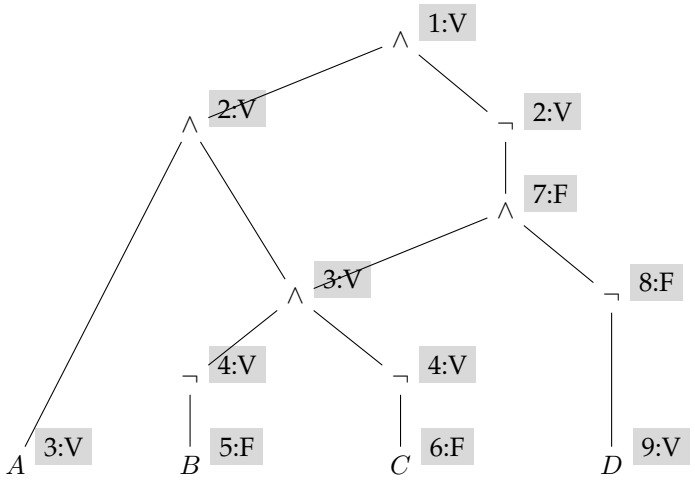
1. Transformar a fórmula inicial para a forma clausal.
2. Estabelecer uma ordem para os símbolos de proposição existentes nas cláusulas.
3. Construir um balde etiquetado para cada símbolo proposicional P, usando a ordenação estabelecida.
4. Adicionar cada cláusula α ao primeiro balde P cuja etiqueta apareça na cláusula.
5. Processar os baldes por ordem, aplicando resolução a todas as combinações de cláusulas que permitam eliminar a etiqueta do balde, e adicionando os resolventes obtidos a partir de cada balde ao primeiro balde que apareça abaixo cuja etiqueta apareça no resolvente.
6. Se a fórmula for satisfazível, é possível determinar uma interpretação que a satisfaça, ou testemunha, fazendo com que a interpretação satisfaça todas as cláusulas de cada balde, de baixo para cima (ou seja, pela ordem inversa à que foi dada).

Nota: Se gerar a cláusula vazia em algum dos passos, a fbf não é satisfazível. Se gerar uma tautologia ($\{A, \ldots, \neg A, \ldots\}$), esta pode ser ignorada.

Exercício 7.1
Considere a seguinte fórmula: $\neg(A \rightarrow (B \vee C)) \wedge ((B \vee C) \vee D)$. Crie o seu DAG, efectue a propagação de marcas de modo a que a fórmula seja verdadeira e apresente uma testemunha.

Resposta:

Em primeiro lugar, é preciso transformar a fórmula para que contenha apenas conjunções e negações:
$\neg(A \rightarrow (B \vee C)) \wedge ((B \vee C) \vee D)$
$\neg\neg(A \wedge \neg(B \vee C)) \wedge \neg(\neg(B \vee C) \wedge \neg D)$
$\neg\neg(A \wedge \neg\neg(\neg B \wedge \neg C)) \wedge \neg(\neg\neg(\neg B \wedge \neg C) \wedge \neg D)$
$(A \wedge (\neg B \wedge \neg C)) \wedge \neg((\neg B \wedge \neg C) \wedge \neg D)$
(notar que $(\neg B \wedge \neg C)$ é comum a ambas as conjunções exteriores)



Esta fórmula é satisfazível e uma testemunha é $I(A) = V$, $I(B) = F$, $I(C) = F$ e $I(D) = V$.

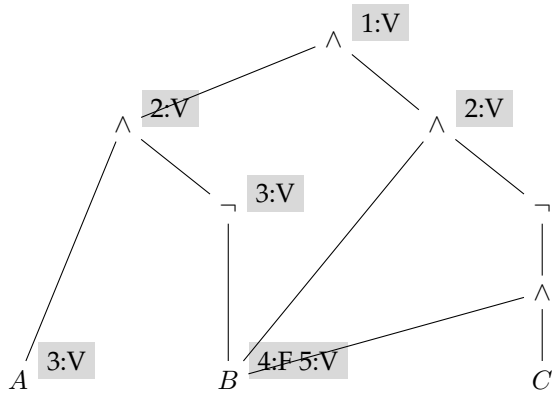
Exercício 7.2
Considere a seguinte fórmula: $(A \wedge \neg B) \wedge (B \wedge \neg(B \wedge C))$.

1. Crie o seu DAG. Efectue a propagação de marcas, de modo a que a fórmula seja verdadeira. O que pode concluir?

2. Faça o mesmo para a negação da fórmula dada. Consegue encontrar uma testemunha usando apenas o algoritmo de propagação de marcas? E usando o algoritmo de teste de nós? O que pode concluir?

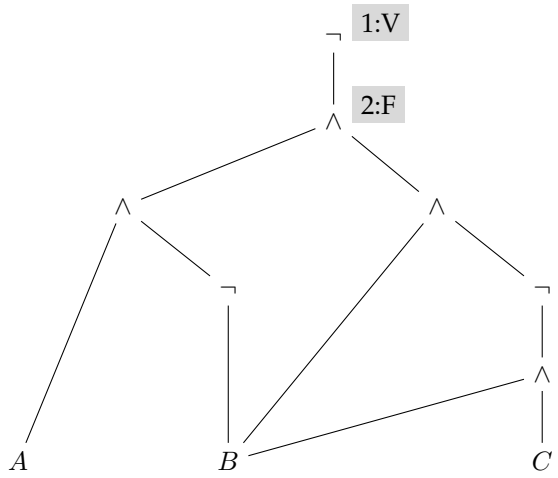
Resposta:

1. A fbf já só tem conjunções e negações, por isso podemos criar já o seu DAG.



Esta fórmula não é satisfazível porque existem duas marcas contraditórias para B, ou seja, a fórmula corresponde a uma contradição.

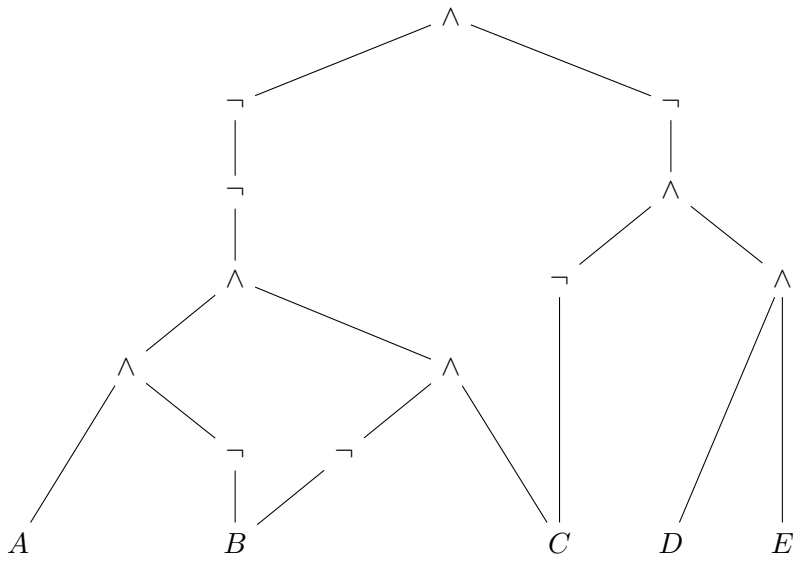
2. Se tivessemos a negação da fórmula inicial, $\neg((A \wedge \neg B) \wedge (B \wedge \neg(B \wedge C)))$, o DAG seria o seguinte. Neste caso, apenas seria possível rotular os dois primeiros nós com o algoritmo de propagação de marcas. Assim, seria necessário usar o algoritmo de teste de nós para tentar encontrar uma testemunha.



Usando o algoritmo de teste de nós para cada um dos nós, com V e com F, vão ficar sempre nós por marcar. Assim, não vai ser possível encontrar uma testemunha para esta fbf.

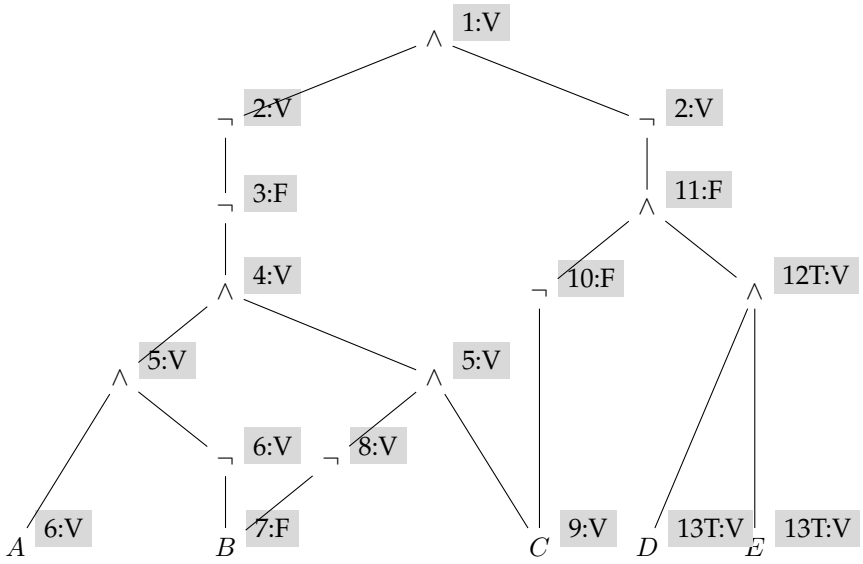
Já agora, convém notar que, se esta fórmula é a negação de uma contradição, então é uma tautologia e tem 8 testemunhas diferentes. Este exemplo mostra que o algoritmo de teste de nós não é completo.

Exercício 7.3
Considere o seguinte DAG. Diga a que fórmula corresponde. Efectue a propagação de marcas, de modo a que a fórmula seja verdadeira. Se for necessário, use o algoritmo de teste de nós. Se for possível, apresente uma testemunha.



Resposta:

Este DAG corresponde à fórmula $\neg\neg((A \wedge \neg B) \wedge (\neg B \wedge C)) \wedge \neg(\neg C \wedge (D \wedge E))$.



Nota: as marcas $12T$, e $13T$ são temporárias, pois são obtidas com o algoritmo de teste de nós, supondo que este algoritmo pode ser usado para testar nós que não sejam folhas. Como todos os nós ficaram marcados com marcas consistentes, o algoritmo termina, pois foi encontrada uma testemunha.

Testemunha: $I(A) = V$, $I(B) = F$, $I(C) = V$, $I(D) = V$, $I(E) = V$.

Exercício 7.4
Considere o seguinte conjunto de cláusulas: $\{\{\neg A, \neg B\}, \{\neg B, A\}, \{B\}\}$. Aplique o algoritmo DP e caso a fórmula seja satisfazível, indique uma testemunha.

1. Usando a ordem $A \prec B$;
2. Usando a ordem $B \prec A$.

Resposta:

1. Usando a ordem $A \prec B$, temos os seguintes baldes iniciais:
   b_A : $\{\neg A, \neg B\}, \{A, \neg B\}$
   b_B : $\{B\}$
   Baldes após aplicação do algoritmo:

b_A :	$\{\neg A, \neg B\}, \{\neg B, A\}$	
b_B :	$\{B\}$	$\{\neg B\}$

   Neste caso, a fórmula correspondente ao conjunto de cláusulas inicial não é satisfazível porque não conseguimos encontrar nenhuma interpretação que satisfaça todas as cláusulas do balde de B.

2. Usando a ordem $B \prec A$, temos os seguintes baldes iniciais:
   b_B : $\{\neg A, \neg B\}, \{\neg B, A\}, \{B\}$
   b_A :
   Baldes após aplicação do algoritmo:

b_B :	$\{\neg A, \neg B\}, \{A, \neg B\}, \{B\}$	
b_A :		$\{\neg A\}, \{A\}$

   Neste caso, a fórmula correspondente ao conjunto de cláusulas inicial não é satisfazível porque não conseguimos encontrar nenhuma interpretação que satisfaça todas as cláusulas do balde de A.

Exercício 7.5
Considere o seguinte conjunto de cláusulas: $\{\{\neg D, B\}, \{\neg C, A\}, \{\neg A, D, C\}, \{\neg C, E\}, \{\neg E\}\}$. Aplique o algoritmo DP usando a ordem $C \prec E \prec D \prec A \prec B$. Caso a fórmula seja satisfazível, indique uma testemunha.

Resposta:

Baldes iniciais:
b_C : $\{\neg C, A\}, \{\neg A, D, C\}, \{\neg C, E\}$
b_E : $\{\neg E\}$
b_D : $\{\neg D, B\}$
b_A :
b_B :

Baldes após aplicação do algoritmo:

b_C :	$\{\neg C, A\}, \{\neg A, D, C\}, \{\neg C, E\}$			
b_E :	$\{\neg E\}$	$\{\neg A, D, E\}$		
b_D :	$\{\neg D, B\}$		$\{\neg A, D\}$	
b_A :				$\{B, \neg A\}$
b_B :				

Notar que a primeira e a segunda cláusulas do balde de C dariam origem ao resolvente $\{A, \neg A, D\}$, que não foi adicionado a nenhum balde porque corresponde a uma tautologia.

Testemunha:

$I(B) = V$ opção

$I(A) = F$ opção

$I(D) = F$ opção

$I(E) = F$ necessariamente por causa de $\{\neg E\}$

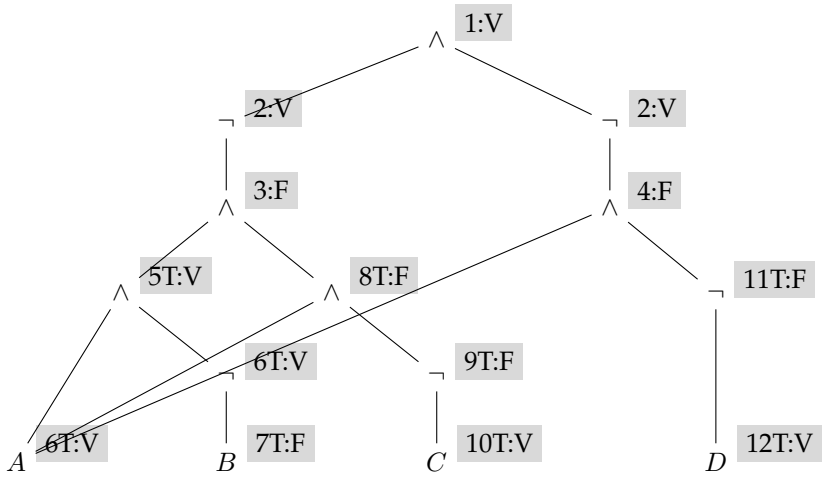
$I(C) = F$ necessariamente por causa de $\{\neg C, A\}$ e $I(A) = F$

Exercício 7.6
Considere a seguinte fórmula $((A \to B) \vee (A \to C)) \wedge (\neg A \vee D)$.
Determine se ela é satisfazível e se for indique uma testemunha, usando:

1. O algoritmo de propagação de marcas (e o algoritmo de teste de nós, se necessário);
2. O algoritmo DP.

Resposta:

1. Primeiro, transformar a fórmula para só conter conjunções e negações:
   $((A \to B) \vee (A \to C)) \wedge (\neg A \vee D)$
   $(\neg(A \wedge \neg B) \vee \neg(A \wedge \neg C)) \wedge \neg(\neg\neg A \wedge \neg D)$
   $\neg(\neg\neg(A \wedge \neg B) \wedge \neg\neg(A \wedge \neg C)) \wedge \neg(\neg\neg A \wedge \neg D)$
   $\neg((A \wedge \neg B) \wedge (A \wedge \neg C)) \wedge \neg(A \wedge \neg D)$



   Marcas temporárias obtidas supondo que o algoritmo de teste de nós pode ser usado para testar nós que não sejam folhas. Como todos os nós ficaram marcados com marcas consistentes, o algoritmo termina, pois foi encontrada uma testemunha.
   Testemunha: $I(A) = V$, $I(B) = F$, $I(C) = V$, $I(D) = V$.

2. Transformar a fórmula para a forma clausal:
   $((A \to B) \vee (A \to C)) \wedge (\neg A \vee D)$
   $((\neg A \vee B) \vee (\neg A \vee C)) \wedge (\neg A \vee D)$
   $\{\{\neg A, B, C\}, \{\neg A, D\}\}$
   Vamos escolher a ordem alfabética para aplicar o algoritmo DP.
   Baldes após aplicação do algoritmo DP:
   b_A : $\{\neg A, B, C\}, \{\neg A, D\}$
   b_B :
   b_C :
   b_D :
   Testemunha:

   $I(D) = V$ opção
   $I(C) = V$ opção
   $I(B) = F$ opção
   $I(A) = V$ opção

   A testemunha encontrada é a mesma que a da alínea anterior, mas poderia não ser. Havia mais alternativas, por exemplo:

$I(D) = V$ opção

$I(C) = F$ opção

$I(B) = F$ opção

$I(A) = F$ necessariamente por causa de $\{\neg A, B, C\}$ e $I(C) = F$ e $I(B) = F$

Podemos ainda dizer que a fórmula é satisfazível porque conseguimos encontrar uma testemunha; é falsificável porque no caso de $I(A) = V$, $I(B) = F$, $I(C) = F$ e $I(D) = F$ a fórmula é falsa; não é tautológica nem contraditória porque é satisfazível e falsificável.

8 Lógica de Primeira Ordem — Sistema de Dedução Natural

Sumário:

- Linguagem da lógica de primeira ordem
- Regras do sistema de dedução natural da lógica de primeira ordem
- Provas usando o sistema de dedução natural da lógica de primeira ordem

Resumo:

Linguagem

As *funções* são caracterizadas pelo seu domínio e contradomínio e usam-se para representar objectos. As funções de aridade zero (sem argumentos) são as *constantes*. Uma *função* representa uma correspondência entre duas classes de entidades. Usa-se quando, dados os seus argumentos, existe um e um só objecto que corresponde ao valor da função.

Cada *variável* pode ter como valor qualquer elemento do domínio da variável, mas tem que ser sempre o mesmo na mesma expressão. No entanto, variáveis diferentes podem ter o mesmo valor.

As *relações* são representadas por predicados, que podem ser *verdadeiros* ou *falsos*. Uma *relação* representa qualquer relação entre duas entidades, e cada objecto pode estar na relação com zero ou mais objectos.

Os *termos* representam objectos sobre os quais queremos falar e correspondem a sintagmas nominais em língua natural. Existem vários tipos de termos: *constantes* (que correspondem a funções de zero argumentos), *variáveis*, e *aplicações de funções* ao número apropriado de termos.

Um termo *chão* é um termo que não contém variáveis livres. Uma fbf que não contém variáveis é uma *fórmula chã*.

Uma *fbf atómica* corresponde apenas a um símbolo de predicado aplicado a termos, ou seja, não tem símbolos lógicos.

Uma ocorrência duma variável diz-se *ligada* numa fbf se esta ocorrência aparecer dentro do domínio do quantificador que a introduz. Uma ocorrência duma variável diz-se *livre* numa fbf se esta não estiver ligada.

Uma fbf que não contém variáveis é uma *fórmula chã*. Uma fbf que não contém variáveis livres é uma *fórmula fechada*.

Uma *substituição* é um conjunto finito de pares ordenados $\{t_1/x_1, \ldots, t_n/x_n\}$ em que cada $x_i (1 \leq i \leq n)$ é uma variável individual e cada $t_i (1 \leq i \leq n)$ é um termo. Numa substituição, todas as variáveis individuais são diferentes e nenhuma das variáveis individuais é igual ao termo correspondente. A cada um dos pares t_i/x_i chama-se uma *ligação*. Uma *substituição chã* é uma substituição na qual nenhum dos termos contém variáveis. A *aplicação da substituição* s a uma fbf α ($\alpha \cdot s$) é obtida a partir de α substituindo todas as ocorrências livres de x_i por t_i. Isto significa que não se substituem variáveis quantificadas.

Se α for uma fbf e t for um termo, dizemos que *t é livre para x em α* se nenhuma ocorrência livre de x em α ocorrer dentro do domínio do quantificador $\forall x$ (ou $\exists x$) em que x é uma variável em t. Informalmente, t ser livre para x em α significa que se todas as ocorrências de x forem substituídas por t, nenhuma ocorrência de uma variável em t deixa de ser livre em $\alpha(t)$. Por exemplo, o termo $g(y, f(b))$ é livre para x na fbf $P(x, y)$, mas não o é na fbf $\forall y[P(x, y)]$. Note-se, por definição, que um termo sem variáveis é sempre livre para qualquer variável em qualquer fbf.

Sistema de Dedução Natural

Usa todas as regras do sistema de dedução natural da lógica proposicional mais as regras de introdução e eliminação de cada um dos quantificadores.

Tal como na lógica proposicional, para simplificar as provas permitimos que se usem algumas das regras básicas com mais graus de liberdade, mas sempre usando fbfs que estejam "no nível de prova correcto".

No fim dos exercícios está um resumo das regras de inferência do sistema de dedução natural da lógica de primeira ordem.

Exercício 8.1
Demonstre os seguintes argumentos, usando o sistema de dedução natural da lógica de primeira ordem:

1. $(\{\forall x[F(x)]\}, \exists x[F(x)])$

2. $(\{\forall x[F(x) \to G(x)], \exists x[F(x) \wedge H(x)]\}, \exists x[G(x) \wedge H(x)])$

Resposta:

1. $(\{\forall x[F(x)]\}, \exists x[F(x)])$

1	$\forall x[F(x)]$	Prem
2	$F(a)$	E$\forall$, 1
3	$\exists x[F(x)]$	I$\exists$, 2

2. $(\{\forall x[F(x) \to G(x)], \exists x[F(x) \wedge H(x)]\}, \exists x[G(x) \wedge H(x)])$

1		$\forall x[F(x) \to G(x)]$	Prem
2		$\exists x[F(x) \wedge H(x)]$	Prem
3	x_0	$F(x_0) \wedge H(x_0)$	Hip
4		$F(x_0)$	E$\wedge$, 3
5		$\forall x[F(x) \to G(x)]$	Rei, 1
6		$F(x_0) \to G(x_0)$	E$\forall$, 5
7		$G(x_0)$	E$\to$, (4, 6)
8		$H(x_0)$	E$\wedge$, 3
9		$G(x_0) \wedge H(x_0)$	I$\wedge$, (7, 8)
10		$\exists x[G(x) \wedge H(x)]$	I$\exists$, 9
11		$\exists x[G(x) \wedge H(x)]$	E$\exists$, (2, (3, 10))

Exercício 8.2
(JPM) Demonstre os seguintes teoremas, usando o sistema de dedução natural da lógica de primeira ordem:

1. $(F(a) \wedge \forall x[F(x) \to G(x)]) \to G(a)$

2. $(\forall x[F(x) \to G(x)] \wedge \forall x[G(x) \to H(x)]) \to \forall x[F(x) \to H(x)]$

3. $(\forall x[F(x) \to H(x)] \wedge \exists y[F(y)]) \to \exists z[H(z)]$

Resposta:

1. $(F(a) \land \forall x[F(x) \to G(x)]) \to G(a)$

1	$F(a) \land \forall x[F(x) \to G(x)]$	Hip
2	$F(a)$	E$\land$, 1
3	$\forall x[F(x) \to G(x)]$	E$\land$, 1
4	$F(a) \to G(a)$	E$\forall$, 3
5	$G(a)$	E$\to$, (2, 4)
6	$(F(a) \land \forall x[F(x) \to G(x)]) \to G(a)$	I$\to$, (1, 5)

2. $(\forall x[F(x) \to G(x)] \land \forall x[G(x) \to H(x)]) \to \forall x[F(x) \to H(x)]$

1		$\forall x[F(x) \to G(x)] \land \forall x[G(x) \to H(x)]$	Hip
2		$\forall x[F(x) \to G(x)]$	E$\land$, 1
3		$\forall x[G(x) \to H(x)]$	E$\land$, 1
4	x_0	$F(x_0)$	Hip
5		$\forall x[F(x) \to G(x)]$	Rei, 2
6		$F(x_0) \to G(x_0)$	E$\forall$, 5
7		$G(x_0)$	E$\to$, (4, 6)
8		$\forall x[G(x) \to H(x)]$	Rei, 3
9		$G(x_0) \to H(x_0)$	E$\forall$, 8
10		$H(x_0)$	E$\to$, (7, 9)
11		$F(x_0) \to H(x_0)$	I$\to$, (4, 10)
12		$\forall x[F(x) \to H(x)]$	I$\forall$, (4, 11)
13		$(\forall x[F(x) \to G(x)] \land \forall x[G(x) \to H(x)]) \to \forall x[F(x) \to H(x)]$	I$\to$, (1, 12)

3. $(\forall x[F(x) \to H(x)] \land \exists y[F(y)]) \to \exists z[H(z)]$

1		$\forall x[F(x) \to H(x)] \land \exists y[F(y)]$	Hip
2		$\forall x[F(x) \to H(x)]$	E$\land$, 1
3		$\exists y[F(y)]$	E$\land$, 1
4	x_0	$F(x_0)$	Hip
5		$\forall x[F(x) \to H(x)]$	Rei, 2
6		$F(x_0) \to H(x_0)$	E$\forall$, 5
7		$H(x_0)$	E$\to$, (4, 6)
8		$\exists z[H(z)]$	I$\exists$, 7
9		$\exists z[H(z)]$	E$\exists$, (3, (4, 8))
10		$(\forall x[F(x) \to H(x)] \land \exists y[F(y)]) \to \exists z[H(z)]$	I$\to$, (1, 9)

Exercício 8.3
Demonstre os seguintes teoremas, que correspondem a aplicações das leis de De Morgan para os quantificadores, usando o sistema de dedução natural da lógica de primeira ordem.

1. $\neg\forall x[F(x)] \rightarrow \exists x[\neg F(x)]$

2. $\exists x[\neg F(x)] \rightarrow \neg\forall x[F(x)]$

3. $\neg\exists x[F(x)] \rightarrow \forall x[\neg F(x)]$

4. $\forall x[\neg F(x)] \rightarrow \neg\exists x[F(x)]$

Resposta:

1. $\neg\forall x[F(x)] \rightarrow \exists x[\neg F(x)]$

1	$\neg\forall x[F(x)]$	Hip
2	$\quad\neg\exists x[\neg F(x)]$	Hip
3	$\quad x_0 \quad\quad \neg F(x_0)$	Hip
4	$\quad\quad\quad \exists x[\neg F(x)]$	I$\exists$, 3
5	$\quad\quad\quad \neg\exists x[\neg F(x)]$	Rei, 2
6	$\quad\quad \neg\neg F(x_0)$	I$\neg$, (3, (4, 5))
7	$\quad\quad F(x_0)$	E$\neg$, 6
8	$\quad \forall x[F(x)]$	I$\forall$, (3, 7)
9	$\quad \neg\forall x[F(x)]$	Rei, 1
10	$\neg\neg\exists x[\neg F(x)]$	I$\neg$, (2, (8, 9))
11	$\exists x[\neg F(x)]$	E$\neg$, 10
12	$\neg\forall x[F(x)] \rightarrow \exists x[\neg F(x)]$	I$\rightarrow$, (1, 11)

2. $\exists x[\neg F(x)] \rightarrow \neg\forall x[F(x)]$

ou

1	$\exists x[\neg F(x)]$	Hip
2	$x_0 \mid \neg F(x_0)$	Hip
3	$\quad \mid \forall x[F(x)]$	Hip
4	$\quad \mid F(x_0)$	E$\forall$, 3
5	$\quad \mid \neg F(x_0)$	Rei, 2
6	$\neg\forall x[F(x)]$	I$\neg$, (3, (4, 5))
7	$\neg\forall x[F(x)]$	E$\exists$, (1, (2, 6))
8	$\exists x[\neg F(x)] \rightarrow \neg\forall x[F(x)]$	I$\rightarrow$, (1, 7)

1	$\exists x[\neg F(x)]$	Hip
2	$\quad \forall x[F(x)]$	Hip
3	$\quad \exists x[\neg F(x)]$	Rei, 1
4	$\quad x_0 \mid \neg F(x_0)$	Hip
5	$\quad \mid \forall x[F(x)]$	Rei, 2
6	$\quad \mid F(x_0)$	E$\forall$, 5
7	$\quad \mid \exists x[\neg F(x)]$	Hip
8	$\quad \mid F(x_0)$	Rei, 6
9	$\quad \mid \neg F(x_0)$	Rei, 4
10	$\quad \neg\exists x[\neg F(x)]$	I$\neg$, (7, (8, 9))
11	$\quad \neg\exists x[\neg F(x)]$	E$\exists$, (3, (4, 10))
12	$\quad \exists x[\neg F(x)]$	Rei, 1
13	$\neg\forall x[F(x)]$	I$\neg$, (2, (12, 11))
14	$\exists x[\neg F(x)] \rightarrow \neg\forall x[F(x)]$	I$\rightarrow$, (1, 13)

3. $\neg\exists x[F(x)] \rightarrow \forall x[\neg F(x)]$

1	$\neg\exists x[F(x)]$	Hip
2	$x_0 \mid \quad \mid F(x_0)$	Hip
3	$\quad \mid \quad \mid \exists x[F(x)]$	I$\exists$, 2
4	$\quad \mid \quad \mid \neg\exists x[F(x)]$	Rei, 1
5	$\quad \mid \neg F(x_0)$	I$\neg$, (2, (3, 4))
6	$\forall x[\neg F(x)]$	I$\forall$, (2, 5)
7	$\neg\exists x[F(x)] \rightarrow \forall x[\neg F(x)]$	I$\rightarrow$, (1, 6)

4. $\forall x[\neg F(x)] \rightarrow \neg\exists x[F(x)]$

1	$\forall x[\neg F(x)]$	Hip
2	$\quad \exists x[F(x)]$	Hip
3	$\quad x_0 \quad F(x_0)$	Hip
4	$\quad\quad\quad \forall x[\neg F(x)]$	Hip
5	$\quad\quad\quad \neg F(x_0)$	E$\forall$, 4
6	$\quad\quad\quad F(x_0)$	Rei, 3
7	$\quad\quad \neg\forall x[\neg F(x)]$	I$\neg$, (4, (5, 6))
8	$\quad \neg\forall x[\neg F(x)]$	E$\exists$, (2, (3, 7))
9	$\quad \forall x[\neg F(x)]$	Rei, 1
10	$\neg\exists x[F(x)]$	I$\neg$, (2, (9, 8))
11	$\forall x[\neg F(x)] \rightarrow \neg\exists x[F(x)]$	I$\rightarrow$, (1, 10)

Lógica de Primeira Ordem — Sistema de Dedução Natural

Prem

n	α	Prem

Hip

n	α	Hip
$n+1$	$\ldots$	

Rep

n	α	
$\vdots$	$\vdots$	
m	α	Rep, n

Rei

n	α	
$\vdots$	$\vdots$	
m	α	Rei, n

I $\rightarrow$

n	α	Hip
$\vdots$	$\vdots$	
m	β	
k	$\alpha \rightarrow \beta$	I$\rightarrow$, (n, m)

E $\rightarrow$

n	α	
$\vdots$	$\vdots$	
m	$\alpha \rightarrow \beta$	
$\vdots$	$\vdots$	
k	β	E$\rightarrow$, (n, m)

I$\wedge$

n	α	
$\vdots$	$\vdots$	
m	β	
$\vdots$	$\vdots$	
k	$\alpha \wedge \beta$	I$\wedge$, (n, m)

E$\wedge$

n	$\alpha \wedge \beta$	
$\vdots$	$\vdots$	
m	α	E$\wedge$, n

ou

n	$\alpha \wedge \beta$	
$\vdots$	$\vdots$	
m	β	E$\wedge$, n

I$\vee$

n	α	
$\vdots$	$\vdots$	
m	$\alpha \vee \beta$	I$\vee$, n

ou

n	α	
$\vdots$	$\vdots$	
m	$\beta \vee \alpha$	I$\vee$, n

E$\vee$

n	$\alpha \vee \beta$		
o		α	Hip
$\vdots$		$\vdots$	
p		γ	
r		β	Hip
$\vdots$		$\vdots$	
s		γ	
m	γ		E$\vee$, (n, (o, p), (r, s))

I$\neg$

n		α	Hip
$\vdots$		$\vdots$	
m		β	
$\vdots$		$\vdots$	
k		$\neg\beta$	
l	$\neg\alpha$		I$\neg$, (n, (m, k))

E$\neg$

n	$\neg\neg\alpha$	
$\vdots$	$\vdots$	
m	α	E$\neg$, n

I$\forall$

n	x_0		
$\vdots$		$\vdots$	
m		$\alpha(x_0)$	
$m+1$	$\forall x[\alpha(x)]$		I$\forall$, (n, m)

E$\forall$

n	$\forall x[\alpha(x)]$	
$\vdots$	$\vdots$	
m	$\alpha(t)$	E$\forall$, n

I$\exists$

n	$\alpha(t)$	
$\vdots$	$\vdots$	
m	$\exists x[\alpha(x)]$	I$\exists$, n

E$\exists$

n	$\exists x[\alpha(x)]$		
m	x_0	$\alpha(x_0)$	Hip
$\vdots$		$\vdots$	
k		β	
$k+1$	β		E$\exists$, (n, (m, k))

9 Lógica de Primeira Ordem — Sistema Semântico

Sumário:

- Sistema semântico da lógica de primeira ordem
- Conceptualização
- Interpretação

Resumo:

Sistema Semântico

O *sistema semântico* especifica em que condições as fbfs da linguagem são verdadeiras ou falsas.

Uma *conceptualização* descreve formalmente uma situação ou um mundo e é um triplo (D, F, R) em que: D é o conjunto das entidades que constituem o mundo sobre o qual vamos falar, o chamado universo de discurso; F é o conjunto das funções que podem ser aplicadas às entidades de D; R é o conjunto das relações ou predicados que podem ser aplicados às entidades de D.

Uma *interpretação* é uma função I cujo domínio são as entidades da linguagem e cujo contradomínio são as entidades da conceptualização. Por a interpretação ser uma função das entidades da linguagem para as entidades da conceptualização, cada constante da linguagem é associada apenas a uma entidade da conceptualização, mas várias constantes da linguagem podem ser associadas à mesma entidade da conceptualização. Sendo α uma entidade da linguagem, é habitual escrever $I(\alpha)$ para representar o resultado da aplicação da interpretação I à entidade α.

Dada uma conceptualização (D, F, R), uma interpretação I é uma função das entidades da linguagem para $D \cup F \cup R$ que deve obedecer às seguintes condições:

1. Cada constante individual f_i^0 é associada com uma entidade do universo de discurso D;
2. Cada letra de função f_i^n é associada a uma função de F. Se f_i^n é uma letra de função com aridade n, correspondendo à função $I(f_i^n)$ da conceptualização, e se $t_1, \ldots, t_n$ são termos, então $f_i^n(t_1, \ldots, t_n)$ corresponde à entidade $I(f_i^n)(I(t_1), \ldots, I(t_n))$ da conceptualização;
3. A cada letra de predicado P_i^n é associada uma relação de R.

Dada uma fbf α, e uma interpretação I, diz-se que I satisfaz α nas seguintes condições:

1. Se α for uma fbf atómica, ou seja, uma fbf da forma $P_i^n(t_1, \ldots, t_n)$, a interpretação I satisfaz α sse o n-tuplo $(I(t_1), \ldots, I(t_n))$ for um elemento da relação $I(P_i^n)$.
2. A interpretação I satisfaz a fbf $\neg\alpha$ sse I não satisfizer α.
3. A interpretação I satisfaz a fbf $\alpha \wedge \beta$ sse I satisfizer as duas fbfs α e β.

4. A interpretação I satisfaz a fbf $\alpha \vee \beta$ sse I satisfizer pelo menos uma das fbfs α ou β.
5. A interpretação I satisfaz a fbf $\alpha \rightarrow \beta$ sse I não satisfizer α ou I satisfizer β.
6. A interpretação I satisfaz a fbf $\forall x[\alpha]$ sse, para toda a substituição $\{a/x\}$, em que "a" é qualquer constante individual, I satisfizer $\alpha \cdot \{a/x\}$.
7. A interpretação I satisfaz a fbf $\exists x[\alpha]$ sse existir uma substituição $\{a/x\}$, em que "a" é uma constante individual, tal que I satisfaz $\alpha \cdot \{a/x\}$.

Uma fbf α *é verdadeira segundo a interpretação* I sse I satisfizer α; caso contrário, α é *falsa segundo a interpretação* I.

Existem quatro tipos de proposições:

- satisfazíveis — as que podem ser ou não ser verdadeiras, dependendo da interpretação escolhida, e são verdadeiras em pelo menos uma interpretação.
- contraditórias — as que nunca podem ser verdadeiras, independentemente da interpretação escolhida.
- tautológicas — as que são sempre verdadeiras, independentemente da interpretação usada.
- falsificáveis — as que podem ser ou não ser verdadeiras, dependendo da interpretação escolhida, e são falsas em pelo menos uma interpretação.

Uma interpretação que satisfaz todas as proposições de um conjunto de proposições chama-se um *modelo* dessas proposições.

A lógica de primeira ordem é *correcta*: se $\{\alpha_1, \ldots, \alpha_n\} \vdash \beta$ então $\{\alpha_1, \ldots, \alpha_n\} \models \beta$.

A lógica de primeira ordem é *completa*: se $\{\alpha_1, \ldots, \alpha_n\} \models \beta$ então $\{\alpha_1, \ldots, \alpha_n\} \vdash \beta$.

Exercício 9.1
Considere a seguinte conceptualização $C = (D, F, R)$:

- $D = \{✦, ✶, ✷, ✢, ✻, ❋\}$
- $F = \{\{(✦, ✢), (✶, ✻), (✷, ❋)\}\}$
- $R = \{\{(✦), (✶), (✷)\}, \{(✢), (✻), (❋)\}, \{(✦, ✶), (✦, ✷), (✶, ✷)\}\}$

e a seguinte interpretação:

- $I(eq) = ✦$
- $I(es) = ✶$
- $I(eo) = ✷$
- $I(fq) = ✢$
- $I(fs) = ✻$
- $I(fo) = ❋$
- $I(florDe) = \{(✦, ✢), (✶, ✻), (✷, ❋)\}$
- $I(Estrela) = \{(✦), (✶), (✷)\}$
- $I(Flor) = \{(✢), (✻), (❋)\}$
- $I(MenosPontas) = \{(✦, ✶), (✦, ✷), (✶, ✷)\}$

Diga, justificando, quais das seguintes fbfs são satisfeitas por esta interpretação para esta conceptualização:

1. $Estrela(eo)$
2. $Estrela(florDe(eq))$
3. $MenosPontas(eo, eq)$
4. $\neg Estrela(es) \vee Flor(fq)$
5. $\exists x[Estrela(x)]$
6. $\forall x, y, z[(MenosPontas(x, y) \wedge MenosPontas(y, z)) \rightarrow MenosPontas(x, z)]$

Resposta:

1. A fbf $Estrela(eo)$ é satisfeita sse $(I(eo)) \in I(Estrela)$, ou seja, sse $(✷) \in \{(✦), (✶), (✷)\}$, o que se verifica e por isso a fbf é satisfeita.
2. A fbf $Estrela(florDe(eq))$ é satisfeita sse $(I(florDe)(I(eq))) \in I(Estrela)$, ou seja, sse $(I(florDe)(✦)) \in I(Estrela)$, ou seja, sse $(✢) \in \{(✦), (✶), (✷)\}$ o que não se verifica e por isso a fbf não é satisfeita.
3. A fbf $MenosPontas(eo, eq)$ é satisfeita sse $(I(eo), I(eq)) \in I(MenosPontas)$, ou seja, sse $(✷, ✦) \in \{(✦, ✶), (✦, ✷), (✶, ✷)\}$, o que não se verifica e por isso a fbf não é satisfeita.
4. A fbf $\neg Estrela(es) \vee Flor(fq)$ é satisfeita sse $Estrela(es)$ não for satisfeita ou $Flor(fq)$ for satisfeita, ou seja, sse $(I(es)) \notin I(Estrela)$ ou $(I(fq)) \in I(Flor)$, ou seja, sse $(✶) \notin \{(✦), (✶), (✷)\}$ ou $(✢) \in \{(✢), (✻), (❋)\}$ a primeira condição não se verifica, mas a segunda verifica-se, logo a disjunção é satisfeita.

5. A fbf $\exists x[Estrela(x)]$ é satisfeita sse existir pelo menos uma substituição $\{a/x\}$, para uma constante a, tal que I satisfaz $Estrela(a)$. Vamos considerar $a = eq$. Neste caso $Estrela(eq)$ é satisfeita sse $(I(eq)) \in I(Estrela)$, ou seja, sse $(✦) \in \{(✦), (✶), (✷)\}$, o que se verifica e por isso a fbf inicial é satisfeita.
6. A fbf $\forall x, y, z[(MenosPontas(x, y) \wedge MenosPontas(y, z)) \rightarrow MenosPontas(x, z)]$ é satisfeita sse para toda a substituição $\{a/x, b/y, c/z\}$, em que a, b, c são constantes individuais, I satisfizer $((MenosPontas(x, y) \wedge MenosPontas(y, z)) \rightarrow MenosPontas(x, z)) \cdot \{a/x, b/y, c/z\}$. Uma implicação é satisfeita sse o seu antecedente não for satisfeito ou o seu consequente for satisfeito. Neste caso, o antecedente nunca é satisfeito a não ser para a substituição $\{eq/x, es/y, eo/z\}$, por isso é necessário saber se I satisfaz $MenosPontas(eq, eo)$, ou seja, se $(I(eq), I(eo)) \in I(MenosPontas)$, ou seja, se $(✦, ✶) \in \{(✦, ✶), (✦, ✷), (✶, ✷)\}$, o que se verifica e por isso a implicação é satisfeita. Como no único caso em que o antecedente é satisfeito o consequente também é, qualquer que seja a substituição considerada, a fbf inicial é satisfeita.

Exercício 9.2
Considere a seguinte conceptualização $C = (D, F, R)$:

- $D = \{I, V, X, C, D, M, 1, 5, 10, 100, 500, 1000\}$
- $F = \{\{(I, 1), (V, 5), (X, 10), (C, 100), (D, 500), (M, 1000)\}\}$
- $R = \{\{(X), (C), (D), (M)\}, \{(I), (V)\}\}$

e a seguinte interpretação:

- $I(um) \mapsto I$
- $I(cinco) \mapsto V$
- $I(dez) \mapsto X$
- $I(cem) \mapsto C$
- $I(quinhentos) \mapsto D$
- $I(mil) \mapsto M$
- $I(valor) \mapsto \{(I, 1), (V, 5), (X, 10), (C, 100), (D, 500), (M, 1000)\}$
- $I(Par) \mapsto \{(X), (C), (D), (M)\}$
- $I(Impar) \mapsto \{(I), (V)\}$

1. Diga, justificando, quais das seguintes fbfs são satisfeitas por esta interpretação para esta conceptualização:
   (a) $Par(cem)$
   (b) $Impar(valor(cinco))$
   (c) $Par(cem) \rightarrow Impar(dez)$
   (d) $\forall x[Par(x)]$
   (e) $\exists x[Impar(valor(x))]$
2. Diga que alterações faria na conceptualização para os resultados serem os intuitivamente esperados. Que mudanças é que isso implicaria na interpretação?
3. Diga, justificando, se o seguinte poderia ser uma interpretação para esta conceptualização. Se não puder, indique todas as razões para não poder.
   - $I(a) \mapsto X$

- $I(a) \mapsto 10$
- $I(b) \mapsto 100$
- $I(c) \mapsto 100$
- $I(d) \mapsto 20$
- $I(f1) \mapsto \{(I, 1), (V, 5), (V, 10), (C, 100)\}$
- $I(R1) \mapsto \{(X), (C), (D)\}$

Resposta:

1. Que fbfs são satisfeitas:
   (a) A fbf $Par(cem)$ é satisfeita sse $(I(cem)) \in I(Par)$, ou seja, sse $(C) \in \{(X), (C), (D), (M)\}$, o que se verifica e por isso a fbf é satisfeita.
   (b) A fbf $Impar(valor(cinco))$ é satisfeita sse $(I(valor)(I(cinco))) \in I(Impar)$, ou seja, sse $(I(valor)(V)) \in I(Impar)$, ou seja, sse $(5) \in \{(I), (V)\}$, o que não se verifica e por isso a fbf não é satisfeita.
   (c) A fbf $Par(cem) \rightarrow Impar(dez)$ é satisfeita sse $Par(cem)$ não for satisfeita ou $Impar(dez)$ for satisfeita, ou seja, sse $(I(cem)) \notin I(Par)$ ou $(I(dez)) \in I(Impar)$, ou seja, sse $(C) \notin \{(X), (C), (D), (M)\}$ ou $(X) \in \{(I), (V)\}$. A primeira condição verifica-se e a segunda não se verifica e por isso a fbf não é satisfeita.
   (d) A fbf $\forall x[Par(x)]$ é satisfeita sse para toda a substituição $\{a/x\}$, em que a é qualquer constante individual, I satisfizer $Par(x) \cdot \{a/x\}$, ou seja, sse $(I(a)) \in I(Par)$ para qualquer constante. Como $(I(um)) \notin I(Par)$, a fbf não é satisfeita.
   (e) A fbf $\exists x[Impar(valor(x))]$ é satisfeita sse existir uma substituição $\{a/x\}$, em que a é uma constante individual, tal que I satisfaz $Impar(valor(a))$. Como o resultado de aplicar a função $valor$ a qualquer constante é um número árabe e não há nenhum tuplo com números árabes em $I(Impar)$, a fbf não é satisfeita.
2. Para os resultados serem os intuitivamente esperados, alterava as interpretações de Par e $Impar$: $I(Par) \mapsto \{(X), (C), (D), (M), (10), (100), (500), (1000)\}$ e $I(Impar) \mapsto \{(I), (V), (1), (5)\}$, fazendo também as correspondentes alterações na conceptualização.
3. Não poderia ser uma interpretação para esta conceptualização por cinco razões:
   - a mesma constante da linguagem (a) tem duas interpretações diferentes, o que não pode acontecer;
   - a constante d tem como valor 20, que não faz parte de D;
   - $f1$ não é uma função, pois tem 2 tuplos em que o primeiro elemento é V;
   - $f1$ não corresponde a nenhum dos elementos da conceptualização;
   - a interpretação de $R1$ não corresponde a nenhuma das relações da conceptualização, pois falta-lhe o último tuplo (M) para corresponder à primeira relação da conceptualização.

Exercício 9.3
Considere a relação binária $EDivisorDe$ no domínio $\{1, 2, 3, 4, 5, 6\}$.

1. Apresente o conjunto de todos os pares que estão na relação para o domínio dado.
2. Diga, justificando, se a relação é: reflexiva, simétrica, ou transitiva.

Resposta:

1. O conjunto de pares é o seguinte:
   $\{(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 2), (2, 4), (2, 6), (3, 3), (3, 6), (4, 4), (5, 5), (6, 6)\}$.

2. Uma relação é reflexiva sse satisfizer $\forall x, y[Rel(x, y) \rightarrow (Rel(x, x) \wedge Rel(y, y))]$. Nota: a definição não poderia ser $\forall x[Rel(x, x)]$, pois era demasiado forte, teria que se aplicar também aos objectos que não fazem parte do domínio da relação; com este antecedente, garante-se que é reflexiva para os objectos do domínio, mas não tem que ser para os outros. A relação do enunciado é reflexiva, pois qualquer número é divisor de si próprio.
Uma relação é simétrica sse satisfizer $\forall x, y[Rel(x, y) \rightarrow Rel(y, x)]$. Esta relação não é simétrica porque é possível encontrar a e b tais que $EDivisorDe(a, b)$ mas $\neg EDivisorDe(a, b)$, por exemplo se $a = 1$ e $b = 2$.
A relação é transitiva pois $\forall x, y, z[(EDivisorDe(x, y) \wedge EDivisorDe(y, z)) \rightarrow EDivisorDe(x, z)]$. Neste caso, podemos ver para todos os casos que tornam o antecedente da implicação verdadeiro, que o consequente também vai ser verdadeiro. Por outras palavras, não conseguimos encontrar um contra-exemplo que torne a definição de transitividade falsa.

Exercício 9.4
Considere a seguinte fbf: $\forall x, y[(A(x) \wedge B(x, y)) \rightarrow A(y)]$. Diga se ela é verdadeira ou falsa para cada uma das seguintes interpretações. Se for falsa, apresente um contra-exemplo.

1. O domínio dos números naturais, onde $A(x)$ é interpretado como "x é par" e $B(x, y)$ é interpretado como "x é igual a y".
2. O domínio dos números naturais, onde $A(x)$ é interpretado como "x é par" e $B(x, y)$ é interpretado como "x é um inteiro divisor de y".
3. O domínio dos números naturais, onde $A(x)$ é interpretado como "x é par" e $B(x, y)$ é interpretado como "x é um inteiro múltiplo de y".
4. O domínio dos booleanos $\{verdadeiro, falso\}$, onde $A(x)$ é interpretado como "x é falso" e $B(x, y)$ é interpretado como "x é igual a y".

Resposta:

1. O domínio dos números naturais, onde $A(x)$ é interpretado como "x é par" e $B(x, y)$ é interpretado como "x é igual a y".
É verdadeira.
2. O domínio dos números naturais, onde $A(x)$ é interpretado como "x é par" e $B(x, y)$ é interpretado como "x é um inteiro divisor de y".
É verdadeira.
3. O domínio dos números naturais, onde $A(x)$ é interpretado como "x é par" e $B(x, y)$ é interpretado como "x é um inteiro múltiplo de y".
É falsa. Por exemplo, $x = 6$ e $y = 3$: 6 é par, é múltiplo de 3, mas 3 não é par.
4. O domínio dos booleanos $\{verdadeiro, falso\}$, onde $A(x)$ é interpretado como "x é falso" e $B(x, y)$ é interpretado como "x é igual a y".
É verdadeira.

Exercício 9.5
Considere a seguinte conceptualização:

- Universo de discurso = {✿, ❀}
- Conjunto de funções = {{(✿, ❀)}, {(✿, ✿), (❀, ✿)}}
- Conjunto de relações = {{(✿, ✿), (✿, ❀), (❀, ❀)}}

e o seguinte conjunto de fórmulas: $\{P_1(f_1(a), b), P_1(f_2(a), b) \rightarrow P_1(b, f_1(a))\}$.

Diga se a seguinte interpretação é modelo deste conjunto de fórmulas:

- $I(a) \mapsto$ ✿
- $I(b) \mapsto$ ❀
- $I(f_1) \mapsto \{(✿, ❀)\}$
- $I(f_2) \mapsto \{(✿, ✿), (❀, ✿)\}$
- $I(P_1) \mapsto \{(✿, ✿), (✿, ❀), (❀, ❀)\}$

Resposta:

Uma interpretação é modelo de um conjunto de fórmulas sse satisfizer todas as fórmulas desse conjunto.

A interpretação satisfaz $P_1(f_1(a), b)$ sse $(I(f_1)(I(a)), I(b)) \in I(P_1)$, ou seja, sse $(I(f_1)(✿), I(b)) \in I(P_1)$, ou seja, sse $(❀, ❀) \in \{(✿, ✿), (✿, ❀), (❀, ❀)\}$, o que se verifica, logo esta fbf é satisfeita por esta interpretação.

A interpretação satisfaz $P_1(f_2(a), b) \rightarrow P_1(b, f_1(a))$ sse não satisfizer $P_1(f_2(a), b)$ ou satisfizer $P_1(b, f_1(a))$. Como $(I(f_2)(I(a)), I(b)) = (✿, ❀) \in \{(✿, ✿), (✿, ❀), (❀, ❀)\}$, satisfaz o antecedente, logo, só satisfaz a implicação se também satisfizer o consequente. Como $(I(b), I(f_1)(I(a))) = (❀, ❀) \in \{(✿, ✿), (✿, ❀), (❀, ❀)\}$, também satisfaz o consequente, logo satisfaz a implicação.

Como a interpretação satisfaz todas as fórmulas do conjunto, é modelo do conjunto.

Exercício 9.6
Considere a seguinte conceptualização $C = (D, F, R)$:

- $D = \{⊞, ⊟, ⊠, ⊡\}$
- $F = \{\{(⊞, ⊠)\}, \{(⊟, ⊡)\}, \{(⊟, ⊡, ⊠), (⊟, ⊠, ⊡)\}\}$
- $R = \{\{(⊞, ⊡)\}, \{(⊠, ⊡)\}, \{(⊞), (⊟), (⊠), (⊡)\}\}$

e a seguinte interpretação:

- $I(a) = ⊞$
- $I(b) = ⊟$
- $I(c) = ⊠$
- $I(d) = ⊡$
- $I(f1) = \{(⊞, ⊠)\}$
- $I(f2) = \{(⊟, ⊡)\}$
- $I(R1) = \{(⊞, ⊡)\}$
- $I(R2) = \{(⊠, ⊡)\}$
- $I(R3) = \{(⊞), (⊟), (⊠), (⊡)\}$

1. Diga, justificando, se esta interpretação para esta conceptualização é um modelo do seguinte conjunto de fórmulas: $\{\exists x, y[R3(x) \wedge R1(x, f2(y))], R2(f1(a), d) \vee \neg R3(b)\}$.

2. Explique porque é que o seguinte não pode ser uma interpretação para esta conceptualização, mencionando todos os erros que foram cometidos.

- $I(a) = \boxplus$
- $I(b) = \boxplus$
- $I(c) = \boxplus$
- $I(c) = \boxminus$
- $I(d) = \circledast$
- $I(f1) = \{(\boxplus, \boxminus), (\boxplus, \boxtimes)\}$
- $I(R1) = \{(\boxplus), (\boxtimes), (\boxdot)\}$

Resposta:

1. Uma interpretação para uma conceptualização é um modelo de um conjunto de fórmulas se satisfizer todas as fórmulas desse conjunto, isto é, se as tornar todas verdadeiras.

   - A fbf $\exists x, y[R3(x) \wedge R1(x, f2(y))]$ é satisfeita sse existir uma substituição $\{c1/x, c2/y\}$, em que $c1$ e $c2$ são constantes individuais, tal que I satisfaz $R3(c1) \wedge R1(c1, f2(c2))$. Sejam $c1 = a$ e $c2 = b$. Neste caso, a fbf $R3(a) \wedge R1(a, f2(b))$ é satisfeita sse $(I(a)) \in I(R3)$ e $(I(a), I(f2)(I(b))) \in I(R1)$, ou seja, sse $(\boxplus) \in \{(\boxplus), (\boxminus), (\boxtimes), (\boxdot)\}$ e $(\boxplus, \boxdot) \in \{(\boxplus, \boxdot)\}$. Como ambas as condições se verificam, a conjunção é satisfeita para esta substituição e a fórmula quantificada existencialmente também é satisfeita.
   - A fbf $R2(f1(a), d) \vee \neg R3(b)$ é satisfeita sse $R2(f1(a), d)$ for satisfeita ou $\neg R3(b)$ for satisfeita. A fbf $R2(f1(a), d)$ é satisfeita sse $(I(f1)(I(a)), I(d)) \in I(R2)$, ou seja, sse $(\boxtimes, \boxdot) \in \{(\boxtimes, \boxdot)\}$, o que se verifica. Como um dos elementos da disjunção é satisfeito, a disjunção inicial é satisfeita.

   Como ambas as fbfs são satisfeitas, esta interpretação para esta conceptualização é modelo deste conjunto de fórmulas.

2. Não pode ser uma interpretação para esta conceptualização porque:

   - A constante c tem duas interpretações diferentes.
   - A interpretação da constante d não pertence ao domínio da conceptualização.
   - A função $f1$ tem dois resultados diferentes quando o seu argumento é $\boxplus$.
   - A interpretação da função $f1$ não pertence ao conjunto de funções da conceptualização.
   - A interpretação da relação $R1$ não pertence ao conjunto de relações da conceptualização.

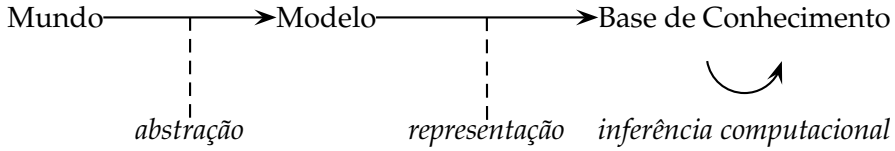
10 Lógica de Primeira Ordem — Representação

Sumário:

- Representação em lógica de primeira ordem

Resumo:

Na **Representação de Conhecimento** a "passagem" do mundo real para a base de conhecimento é feita por seres humanos. Apenas a inferência é feita pelos computadores.



Do ponto de vista da Representação de Conhecimento, as fbfs que interessam são as satisfazíveis e falsificáveis, pois são as que permitem separar os mundos que as satisfazem dos que não as satisfazem, e assim ajudar a definir o mundo que nos interessa. As tautológicas são usadas como passos intermédios nas provas, servem para fazer deduções. As contraditórias não interessam nunca, pois nunca podem ser verdadeiras em nenhum mundo.

Na Representação de Conhecimento em lógica:

- As *funções* usam-se para representar objectos do domínio.
  - As funções de aridade zero chamam-se *constantes* e representam objectos individuais. Ex: $Dumbo$, $Nemo$.
  - As funções de aridade maior que zero representam correspondências entre entidades, que são obrigatoriamente únicas. Ex: $caudaDe(Dumbo)$, $paiDe(Nemo)$. Não se poderia usar apenas uma função para representar uma das patas do Dumbo; ou usávamos uma função para cada pata ou uma função que tivesse como valor todas as patas do Dumbo.
- As *variáveis* representam objectos do domínio e devem estar quantificadas existencialmente ou universalmente.
- Os *predicados* representam relações entre objectos ou classes de objectos e podem ser verdadeiros ou falsos. Ex: $Elefante(Dumbo)$, $Peixe(Nemo)$, $Maior(Dumbo, Nemo)$.

Alguns documentos com exercícios e resumos sobre representação do conhecimento:

- Exercises for the practical classes of Knowledge Representation and Reasoning: `http://web.ist.utl.pt/acardoso/docs/2012-rcr-guiaPraticas.pdf`
- Exercícios para as aulas práticas de Representação do Conhecimento `http://web.ist.utl.pt/acardoso/docs/2007-rc-guiaPraticas.pdf`
- Soluções de alguns dos exercícios de Exercícios de Representação do Conhecimento - Vol I `http://web.ist.utl.pt/acardoso/docs/2007-rc-exerciciosResolvidos.pdf`

Exercício 10.1
Represente em lógica de primeira ordem cada uma das seguintes frases:

1. O Miau é um gato castanho.
2. Os gatos são animais.
3. Nenhum gato é um cão.
4. Nem todos os gatos gostam de leite.
5. Os gatos não gostam de cães.
   (Nenhum gato gosta de nenhum cão.)
6. Existe um cão de quem todos os gatos gostam.
   (Todos os gatos gostam do mesmo cão.)
7. Todos os gatos gostam de algum cão.
   (Pode ser um cão diferente para cada gato.)
8. Se algum gato gostar do Rui, então o Miau também gosta do Rui.
9. O Miau é um siamês ou um bobtail, mas não os dois simultaneamente.
   (Não pode usar o ou exclusivo.)
10. A cauda do Miau é comprida.
11. Todos os gatos têm cauda.

Resposta:

1. O Miau é um gato castanho.
   $Gato(Miau) \wedge Castanho(Miau)$
2. Os gatos são animais.
   $\forall x[Gato(x) \rightarrow Animal(x)]$
3. Nenhum gato é um cão.
   $\forall x[Gato(x) \rightarrow \neg Cao(x)]$
   ou
   $\neg(\exists x[Gato(x) \wedge Cao(x)])$
4. Nem todos os gatos gostam de leite.
   $\exists x[Gato(x) \wedge \neg Gosta(x, Leite)]$
   ou
   $\neg\forall x[Gato(x) \rightarrow Gosta(x, Leite)]$
5. Os gatos não gostam de cães.
   $\forall x, y[(Gato(x) \wedge Cao(y)) \rightarrow \neg Gosta(x, y)]$
   ou
   $\neg\exists x, y[Gato(x) \wedge Cao(y) \wedge Gosta(x, y)]$
6. Existe um cão de quem todos os gatos gostam.
   $\exists x[Cao(x) \wedge \forall y[Gato(y) \rightarrow Gosta(y, x)]]$
7. Todos os gatos gostam de algum cão.
   $\forall x[Gato(x) \rightarrow \exists y[Cao(y) \wedge Gosta(x, y)]]$
8. Se algum gato gostar do Rui, então o Miau também gosta do Rui.
   $\exists x[Gato(x) \wedge Gosta(x, Rui)] \rightarrow Gosta(Miau, Rui)$

9. O Miau é um siamês ou um bobtail, mas não os dois simultaneamente.
   $(Siames(Miau) \vee Bobtail(Miau)) \wedge \neg(Siames(Miau) \wedge Bobtail(Miau))$ ou
   $(Siames(Miau) \wedge \neg Bobtail(Miau)) \vee (\neg Siames(Miau) \wedge Bobtail(Miau))$
   Se representar usando implicações, atenção à contrapositiva.
   Ver a discussão da solução do exercício 2.3.1 dos exercícios resolvidos de Representação do Conhecimento, acerca do BolaDeNeve, que é cão ou gato, mas não as duas coisas simultaneamente.
10. A cauda do Miau é comprida.
    $Comprida(caudaDe(Miau))$
11. Todos os gatos têm cauda.
    $\forall x[Gato(x) \rightarrow Tem(x, caudaDe(x))]$

Exercício 10.2
Suponha que $N(x)$ representa que "x é um número", $P(x)$ representa que "x é par", $I(x)$ representa que "x é ímpar" e $M(x, y)$ representa que "x é maior que y". Traduza as seguintes fbfs para linguagem comum. Se alguma delas não fizer sentido, explique porquê.

1. $\exists x[N(x)]$
2. $\forall x[I(x) \rightarrow N(x)]$
3. $\neg\forall x[N(x) \rightarrow P(x)]$
4. $\exists x[N(x) \wedge \neg P(x)]$
5. $\forall x[I(x) \rightarrow \neg P(x)]$
6. $\forall x[P(x) \rightarrow \exists y[I(y) \wedge M(y, x)]]$
7. $\exists x[P(x) \wedge \forall y[(P(y) \wedge x \neq y) \rightarrow M(y, x)]]$
8. $\exists x[P(x) \rightarrow N(x)]$

Resposta:

1. $\exists x[N(x)]$
   Existe pelo menos um número.
2. $\forall x[I(x) \rightarrow N(x)]$
   Todos os ímpares são números.
3. $\neg\forall x[N(x) \rightarrow P(x)]$
   Nem todos os números são pares.
4. $\exists x[N(x) \wedge \neg P(x)]$
   Existe pelo menos um número que não é par, ou nem todos os números são pares, como a anterior.
5. $\forall x[I(x) \rightarrow \neg P(x)]$
   Os ímpares não são pares.
6. $\forall x[P(x) \rightarrow \exists y[I(y) \wedge M(y, x)]]$
   Para qualquer par, existe pelo menos um ímpar que é maior do que ele.
7. $\exists x[P(x) \wedge \forall y[(P(y) \wedge x \neq y) \rightarrow M(y, x)]]$
   Existe um par que é menor do que todos os outros pares.

8. $\exists x[P(x) \rightarrow N(x)]$
   Esta proposição não tem utilidade, pois basta existir um elemento do domínio que não satisfaça o predicado $P(x)$ para ela ser verdadeira. Está aqui porque é um erro que aparece muito frequentemente nos testes.

11 Lógica de Primeira Ordem — Resolução

Sumário:

- Unificação e resolução em lógica de primeira ordem

Resumo:

Passos da passagem para a forma clausal na LPO:

1. Eliminação de $\rightarrow$: substituir todas as ocorrências de $\alpha \rightarrow \beta$ por $\neg\alpha \vee \beta$.
2. Redução do domínio de $\neg$: eliminação da dupla negação $\neg\neg\alpha \leftrightarrow \alpha$, utilização das leis de De Morgan $\neg(\alpha \vee \beta) \leftrightarrow (\neg\alpha \wedge \neg\beta)$, $\neg(\alpha \wedge \beta) \leftrightarrow (\neg\alpha \vee \neg\beta)$ e utilização das segundas leis de De Morgan $\neg\forall x[\alpha(x)] \leftrightarrow \exists x[\neg\alpha(x)]$ e $\neg\exists x[\alpha(x)] \leftrightarrow \forall x[\neg\alpha(x)]$.
3. Normalização de variáveis: mudar o nome de algumas das variáveis de modo a que cada ocorrência de um quantificador esteja associada a um único nome de variável.
4. Eliminação dos quantificadores existenciais: substituir todas as variáveis quantificadas existencialmente por constantes ou funções de Skolem, consoante não estejam ou estejam no âmbito de quantificadores universais.
5. Conversão para a forma "Prenex" normal: mover todas as ocorrências de quantificadores universais para a esquerda da fbf.
6. Eliminação da quantificação universal: uma vez que as fbfs de origem não tinham variáveis livres, todas as variáveis existentes na fbf após os passos anteriores são quantificadas universalmente, e como a ordem por que aparecem os quantificadores universais não é importante, podemos eliminar a ocorrência explícita dos quantificadores universais e assumir que todas as variáveis são quantificadas universalmente.
7. Obtenção da forma conjuntiva normal: transformar a fórmula numa conjunção de disjunções usando a propriedade distributiva da disjunção $\alpha \vee (\beta \wedge \gamma) \leftrightarrow (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$.
8. Eliminação de $\wedge$: transformar a fórmula num conjunto de cláusulas (ou disjunções).
9. Eliminação de $\vee$: transformar cada cláusula num conjunto de literais.

Uma *substituição* é um conjunto finito de pares ordenados $\{t_1/x_1, \ldots, t_n/x_n\}$ em que cada x_i é uma variável individual e cada t_i é um termo. Numa substituição, todas as variáveis individuais são diferentes e nenhuma das variáveis individuais é igual ao termo correspondente.

A *unificação* é o processo que permite determinar se duas fbfs atómicas podem ser tornadas iguais através de substituições apropriadas para as suas variáveis livres.

Sendo $s_1 = \{t_1/x_1, \ldots, t_n/x_n\}$ e $s_2 = \{u_1/y_1, \ldots, u_m/y_m\}$ duas substituições, a *composição das substituições* s_1 e s_2 , representada por $s_1 \circ s_2$, é a substituição s tal que para qualquer fbf $\alpha, \alpha \cdot s = (\alpha \cdot s_1) \cdot s_2$. A composição das substituições s_1 e s_2 obtém-se aplicando s_2

aos termos de s_1 e adicionando a s_1 os pares de s_2 que contêm variáveis que não ocorrem em s_1. Ou seja, $s_1 \circ s_2$ é obtido do conjunto $\{t_1 \cdot s_2/x_1, \ldots, t_n \cdot s_2/x_n, u_1/y_1, \ldots, u_m/u_m\}$, removendo todos os elementos $t_i \cdot s_2/x_i$ tais que $t_i \cdot s_2 = x_i$ e todos os elementos u_j/y_j tais que $y_j \in \{x_1, \ldots, x_n\}$. A composição de substituições é associativa mas não é comutativa.

Um conjunto de fbfs atómicas $\{\alpha_1, \ldots, \alpha_m\}$ diz-se *unificável* sse existir uma substituição s que torna idênticas todas as fbfs do conjunto, ou seja, sse existir uma substituição s tal que $\alpha_1 \cdot s = \ldots = \alpha_m \cdot s$. Neste caso, a substituição s diz-se o *unificador* do conjunto $\{\alpha_1, \ldots, \alpha_m\}$. Se s é um unificador do conjunto $\{\alpha_1, \ldots, \alpha_m\}$, então $\{\alpha_1 \cdot s, \ldots, \alpha_m \cdot s\} = \{\alpha_1 \cdot s\}$.

Dado um conjunto de fbfs atómicas $\{\alpha_1, \ldots, \alpha_m\}$, define-se o *unificador mais geral* ou *mgu* deste conjunto como sendo um unificador, s, de $\{\alpha_1, \ldots, \alpha_m\}$, com a seguinte propriedade: se s_1 for um unificador de $\{\alpha_1, \ldots, \alpha_m\}$, então existe uma substituição s_2 tal que $s = s_1 \circ s_2$. Uma propriedade importante do unificador mais geral é o facto de ser único (excepto para variantes alfabéticas de variáveis).

Resolução com cláusulas com variáveis Sejam Ψ e Φ duas cláusulas, α e β dois literais tais que $\alpha \in \Psi$ e $\neg\beta \in \Phi$, e α e β são unificáveis. Seja s o unificador mais geral de α e β. Então, usando o princípio da resolução, podemos inferir a cláusula $((\Psi - \{\alpha\}) \cup (\Phi - \{\neg\beta\})) \cdot s$. A cláusula obtida é chamada o resolvente das cláusulas Ψ e Φ.

Com a utilização de cláusulas com variáveis, a resolução pode ser utilizada para responder a dois tipos de questões, questões do tipo "verdadeiro ou falso" e questões do tipo "quem ou qual". As questões do tipo verdadeiro ou falso pretendem saber se uma dada cláusula pode ser derivada de um conjunto de cláusulas. Nas questões do tipo quem ou qual não estamos interessados em saber se uma dada proposição é consequência de um conjunto de cláusulas, mas sim quais são as instâncias que fazem com que uma fbf que contém variáveis livres seja consequência de um conjunto de cláusulas. De modo a fornecer uma resposta a esta questão, utilizando resolução, para além de necessitarmos de transformar as premissas para a forma clausal, teremos que adicionar uma nova fbf que especifica quais são as respostas desejadas.

Tal como na lógica proposicional, a resolução é correcta mas não é completa, no sentido em que não existe a garantia de derivar todas as cláusulas que são consequências lógicas de um conjunto de cláusulas. No entanto, a resolução é completa quanto à refutação. Por esta razão é que normalmente as provas em resolução são feitas por redução ao absurdo.

Exercício 11.1
Utilize o algoritmo de unificação para determinar quais dos seguintes conjuntos de fbfs são unificáveis, e, no caso de o serem, determine o unificador mais geral. Mostre todos os passos intermédios usados nos cálculos.

1. $\{P(a,x,x), P(a,b,c)\}$
2. $\{P(a,x,f(x)), P(x,y,z)\}$
3. $\{P(x,y), Q(x,y)\}$
4. $\{Colocou(x_1, SenhorAneis, y_1), Colocou(Maria, x_2, topo(y2)),$
   $Colocou(x_3, SenhorAneis, topo(MesaAzul))\}$

Resposta:

1. $\{P(a,x,x), P(a,b,c)\}$

Conjunto de fbfs	Conj. Desacordo	Substituição
$\{P(a,x,x), P(a,b,c)\}$	$\{x,b\}$	$\{b/x\}$
$\{P(a,b,b), P(a,b,c)\}$	$\{b,c\}$	não unificáveis

2. $\{P(a,x,f(x)), P(x,y,z)\}$

Conjunto de fbfs	Conj. Desacordo	Substituição
$\{P(a,x,f(x)), P(x,y,z)\}$	$\{a,x\}$	$\{a/x\}$
$\{P(a,a,f(a)), P(a,y,z)\}$	$\{a,y\}$	$\{a/y\}$
$\{P(a,a,f(a)), P(a,a,z)\}$	$\{f(a),z\}$	$\{f(a)/z\}$
$\{P(a,a,f(a))\}$		

O unificador mais geral é $\{f(a)/z, a/y, a/x\}$.

3. $\{P(x,y), Q(x,y)\}$

Conjunto de fbfs	Conj. Desacordo	Substituição
$\{P(x,y), Q(x,y)\}$	$\{P,Q\}$	não unificáveis

4. $\{Colocou(x_1, SenhorAneis, y_1), Colocou(Maria, x_2, topo(y2)), Colocou(x_3, SenhorAneis, topo(MesaAzul))\}$

Conjunto de fbfs	Conj. Desacordo	Substituição
$\{C(x_1, SA, y_1), C(M, x_2, t(y2)), C(x_3, SA, t(MA))\}$	$\{x_1, M, x_3\}$	$\{M/x_1\}, \{M/x_3\}$
$\{C(M, SA, y_1), C(M, x_2, t(y2)), C(M, SA, t(MA))\}$	$\{SA, x_2\}$	$\{SA/x_2\}$
$\{C(M, SA, y_1), C(M, SA, t(y2)), C(M, SA, t(MA))\}$	$\{y_1, t(y_2), t(MA)\}$	$\{t(y_2)/y_1\}$
$\{C(M, SA, t(y_2)), C(M, SA, t(MA))\}$	$\{y_2, MA\}$	$\{MA/y_2\}$
$\{C(M, SA, t(MA))\}$		

O unificador mais geral é $\{MA/y_2, t(MA)/y_1, SA/x_2, M/x_1, M/x_3\}$.
Nota1: Na primeira linha da tabela foram aplicados dois passos do algoritmo de unificação.
Nota2: No UMG é $t(MA)/y_1$ e não $t(y_2)/y_1$ por causa da composição de substituições.

Uma forma de verificar se o unificador mais geral está bem calculado é saber que quando se aplica essa substituição a cada uma das fbfs do conjunto inicial deve ter como resultado, em apenas um passo, a fbf do conjunto final.

Exercício 11.2
Passe as seguintes *fbfs* da lógica de primeira ordem para a forma clausal.

1. $\exists x[A(x)] \wedge \forall x, y[B(x) \vee \exists w[C(x,y,w)] \vee \exists w[D(w,y)]] \wedge \exists x[E(x) \vee F(x)]$
2. $\forall x[(A(x) \wedge \exists y[B(y) \wedge C(x,y)]) \rightarrow D(x)]$

3. $\forall x[A(x) \rightarrow \exists y[B(x,y) \wedge C(y)]] \wedge \exists x[D(x)]$

Resposta:

1. $\exists x[A(x)] \wedge \forall x,y[B(x) \vee \exists w[C(x,y,w)] \vee \exists w[D(w,y)]] \wedge \exists x[E(x) \vee F(x)]$
   (a) Eliminação de $\rightarrow$: já está.
   (b) Redução do domínio de $\neg$: já está.
   (c) Normalização de variáveis: $\exists x[A(x)] \wedge \forall y,z[B(y) \vee \exists w[C(y,z,w)] \vee \exists v[D(v,z)]] \wedge \exists u[E(u) \vee F(u)]$
   (d) Eliminação de $\exists$: $A(sk_1) \wedge \forall y,z[B(y) \vee C(y,z,skf_1(y,z)) \vee D(skf_2(y,z),z)] \wedge (E(sk_2) \vee F(sk_2))$
   (e) Mover $\forall$ para esquerda: $\forall y,z[A(sk_1) \wedge (B(y) \vee C(y,z,skf_1(y,z)) \vee D(skf_2(y,z),z)) \wedge (E(sk_2) \vee F(sk_2))]$
   (f) Eliminação de $\forall$: $A(sk_1) \wedge (B(y) \vee C(y,z,skf_1(y,z)) \vee D(skf_2(y,z),z)) \wedge (E(sk_2) \vee F(sk_2))$
   (g) Obtenção da forma conjuntiva normal: já está.
   (h) Eliminação de $\wedge$: $\{A(sk_1), B(y) \vee C(y,z,skf_1(y,z)) \vee D(skf_2(y,z),z), E(sk_2) \vee F(sk_2)\}$
   (i) Eliminação de $\vee$: $\{\{A(sk_1)\}, \{B(y), C(y,z,skf_1(y,z)), D(skf_2(y,z),z)\}, \{E(sk_2), F(sk_2)\}\}$
2. $\forall x[(A(x) \wedge \exists y[B(y) \wedge C(x,y)]) \rightarrow D(x)]$
   (a) Eliminação de $\rightarrow$: $\forall x[\neg(A(x) \wedge \exists y[B(y) \wedge C(x,y)]) \vee D(x)]$
   (b) Redução do domínio de $\neg$: $\forall x[\neg A(x) \vee \neg\exists y[B(y) \wedge C(x,y)] \vee D(x)]$
   $\forall x[\neg A(x) \vee \forall y[\neg B(y) \vee \neg C(x,y)] \vee D(x)]$
   (c) Normalização de variáveis: já está.
   (d) Eliminação de $\exists$: já está.
   (e) Mover $\forall$ para esquerda: $\forall x,y[\neg A(x) \vee \neg B(y) \vee \neg C(x,y) \vee D(x)]$
   (f) Eliminação de $\forall$: $\neg A(x) \vee \neg B(y) \vee \neg C(x,y) \vee D(x)$
   (g) Obtenção da forma conjuntiva normal: já está.
   (h) Eliminação de $\wedge$: $\{\neg A(x) \vee \neg B(y) \vee \neg C(x,y) \vee D(x)\}$
   (i) Eliminação de $\vee$: $\{\{\neg A(x), \neg B(y), \neg C(x,y), D(x)\}\}$
3. $\forall x[A(x) \rightarrow \exists y[B(x,y) \wedge C(y)]] \wedge \exists x[D(x)]$
   (a) Eliminação de $\rightarrow$: $\forall x[\neg A(x) \vee \exists y[B(x,y) \wedge C(y)]] \wedge \exists x[D(x)]$
   (b) Redução do domínio de $\neg$: já está.
   (c) Normalização de variáveis: $\forall x[\neg A(x) \vee \exists y[B(x,y) \wedge C(y)]] \wedge \exists z[D(z)]$
   (d) Eliminação de $\exists$: $\forall x[\neg A(x) \vee (B(x,skf_1(x)) \wedge C(skf_1(x)))] \wedge D(sk_1)$
   (e) Mover $\forall$ para esquerda: $\forall x[(\neg A(x) \vee (B(x,skf_1(x)) \wedge C(skf_1(x)))) \wedge D(sk_1)]$
   (f) Eliminação de $\forall$: $(\neg A(x) \vee (B(x,skf_1(x)) \wedge C(skf_1(x)))) \wedge D(sk_1)$
   (g) Obtenção da forma conjuntiva normal: $(\neg A(x) \vee B(x,skf_1(x))) \wedge (\neg A(x) \vee C(skf_1(x))) \wedge D(sk_1)$
   (h) Eliminação de $\wedge$: $\{\neg A(x) \vee B(x,skf_1(x)), \neg A(x) \vee C(skf_1(x)), D(sk_1)\}$
   (i) Eliminação de $\vee$: $\{\{\neg A(x), B(x,skf_1(x))\}, \{\neg A(x), C(skf_1(x))\}, \{D(sk_1)\}\}$

Exercício 11.3
Considere o seguinte conjunto de cláusulas: $\{\{\neg A(x), B(x), C(x)\}, \{A(x)\}, \{\neg B(a)\}, \{\neg A(y), \neg C(y)\}\}$.

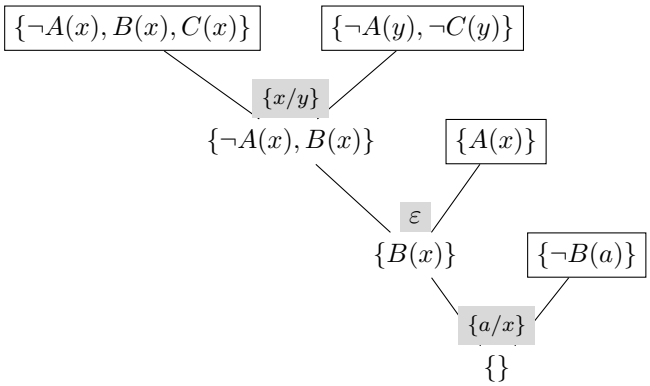
1. Apresente uma demonstração por refutação a partir desse conjunto.

2. Apresente uma demonstração por refutação a partir desse conjunto, usando resolução unitária.

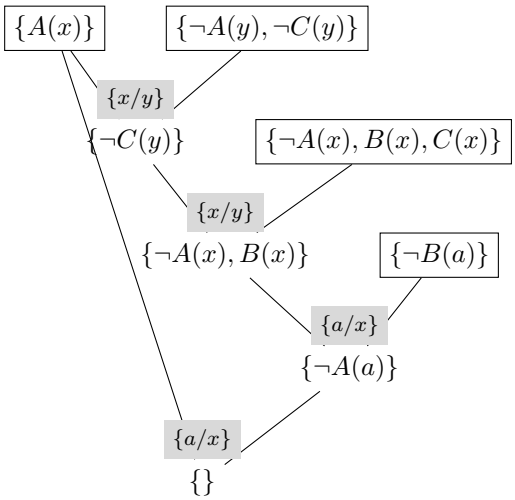
3. Apresente uma demonstração por refutação a partir desse conjunto, usando resolução linear e $\{\neg B(a)\}$ como cláusula inicial.

Resposta:

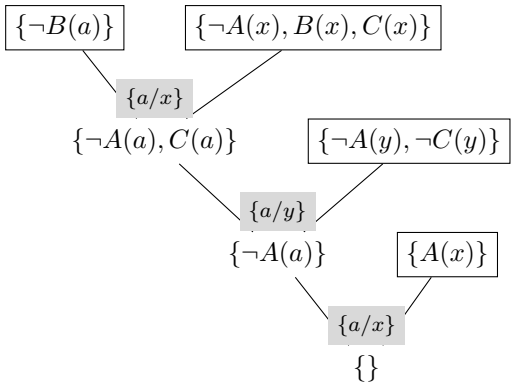
1. Demonstração por refutação a partir do conjunto:
Uma demonstração por refutação é uma demonstração de $\{\}$ a partir do conjunto de cláusulas dado.



2. Demonstração por refutação a partir do conjunto, usando resolução unitária:
Na resolução unitária usa-se sempre pelo menos uma cláusula unitária (ou seja, só com um literal). Neste caso, também é resolução linear.



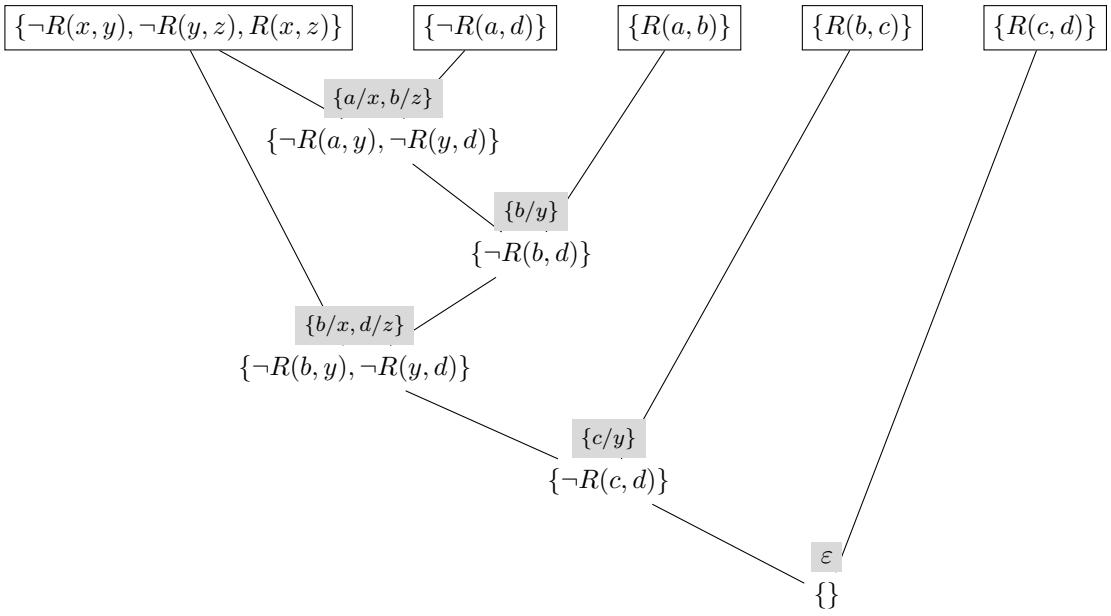
3. Demonstração por refutação a partir do conjunto, usando resolução linear e $\{\neg B(a)\}$ como cláusula inicial:
A resolução linear começa pela cláusula inicial e depois usa sempre um sucessor dessa cláusula nos passos subsequentes.



Exercício 11.4
Demonstre, usando resolução, o seguinte argumento:
$(\{\forall x, y, z[(R(x, y) \wedge R(y, z)) \rightarrow R(x, z)], R(a, b), R(b, c), R(c, d)\}, R(a, d))$.

Resposta:

Em primeiro lugar, é necessário passar as premissas e a negação da conclusão para a forma clausal. Ficamos com o seguinte conjunto de cláusulas:
$\{\{\neg R(x, y), \neg R(y, z), R(x, z)\}, \{R(a, b)\}, \{R(b, c)\}, \{R(c, d)\}, \{\neg R(a, d)\}\}$.
Depois, é necessário aplicar resolução até chegar a uma contradição, preferencialmente começando com a negação da conclusão.



Exercício 11.5
Demonstre os argumentos do exercício 8.1, usando resolução.

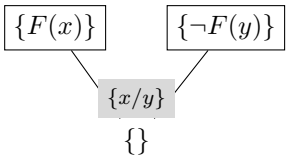
Resposta:

Em cada caso, estamos a tentar provar que a conclusão é derivável a partir das premissas do argumento. Cada premissa (individualmente) e a conclusão negada são passadas para a forma clausal e tenta-se chegar a uma contradição (representada pela cláusula vazia). Se conseguirmos, é porque é impossível todas as premissas serem verdadeiras e a conclusão falsa, e por isso o argumento é válido, o que significa que a conclusão é derivável a partir das premissas.

1. $(\{\forall x[F(x)]\}, \exists x[F(x)])$
   Passagem para a forma clausal de $\forall x[F(x)], \neg\exists x[F(x)]$:

   (a) Eliminação de $\rightarrow$: já está.
   (b) Redução do domínio de $\neg$: $\forall x[F(x)], \forall x[\neg F(x)]$
   (c) Normalização de variáveis: $\forall x[F(x)], \forall y[\neg F(y)]$
   (d) Eliminação de $\exists$: já está.
   (e) Mover $\forall$ para esquerda: já está.
   (f) Eliminação de $\forall$: $F(x), \neg F(y)$
   (g) Obtenção da forma conjuntiva normal: já está.
   (h) Eliminação de $\wedge$: $\{F(x), \neg F(y)\}$
   (i) Eliminação de $\vee$: $\{\{F(x)\}, \{\neg F(y)\}\}$

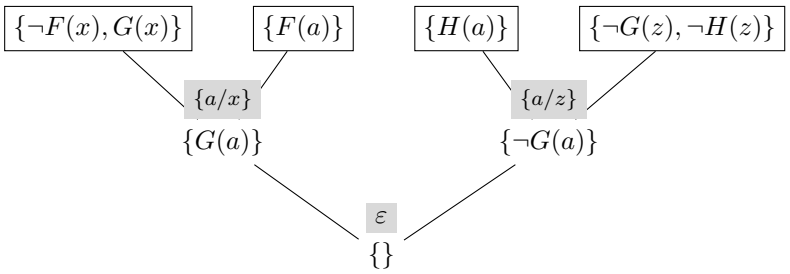
   Aplicação do princípio da resolução:



2. $(\{\forall x[F(x) \rightarrow G(x)], \exists x[F(x) \wedge H(x)]\}, \exists x[G(x) \wedge H(x)])$
   Passagem para a forma clausal de $\forall x[F(x) \rightarrow G(x)], \exists x[F(x) \wedge H(x)], \neg\exists x[G(x) \wedge H(x)]$:

   (a) Eliminação de $\rightarrow$: $\forall x[\neg F(x) \vee G(x)], \exists x[F(x) \wedge H(x)], \neg\exists x[G(x) \wedge H(x)]$
   (b) Redução do domínio de $\neg$: $\forall x[\neg F(x) \vee G(x)], \exists x[F(x) \wedge H(x)], \forall x[\neg G(x) \vee \neg H(x)]$
   (c) Normalização de variáveis: $\forall x[\neg F(x) \vee G(x)], \exists y[F(y) \wedge H(y)], \forall z[\neg G(z) \vee \neg H(z)]$
   (d) Eliminação de $\exists$: $\forall x[\neg F(x) \vee G(x)], F(a) \wedge H(a), \forall z[\neg G(z) \vee \neg H(z)]$
   (e) Mover $\forall$ para esquerda: já está.
   (f) Eliminação de $\forall$: $\{\neg F(x) \vee G(x)\}, \{F(a) \wedge H(a)\}, \{\neg G(z) \vee \neg H(z)\}$
   (g) Obtenção da forma conjuntiva normal: já está.
   (h) Eliminação de $\wedge$: $\{\neg F(x) \vee G(x), \{F(a), H(a), \neg G(z) \vee \neg H(z)\}$
   (i) Eliminação de $\vee$: $\{\{\neg F(x), G(x)\}, \{F(a)\}, \{H(a)\}, \{\neg G(z), \neg H(z)\}\}$

   Aplicação do princípio da resolução:



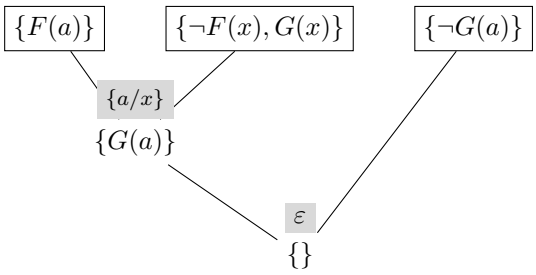
Exercício 11.6
Demonstre os teoremas do exercício 8.2, usando resolução.

Resposta:

Em cada caso, estamos a tentar provar que uma fórmula é teorema. Negamo-la, passamo-la para a forma clausal, e tentamos chegar a uma contradição (representada pela cláusula vazia) usando resolução. Se conseguirmos, é porque é impossível a fórmula ser falsa, e por isso tem que ser sempre verdadeira, ou seja, é um teorema.

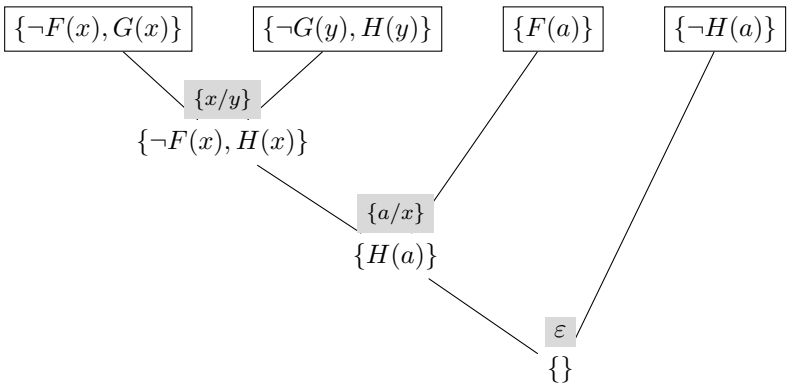
1. $(F(a) \land \forall x[F(x) \to G(x)]) \to G(a)$
   Passagem para a forma clausal de $\neg((F(a) \land \forall x[F(x) \to G(x)]) \to G(a))$:
   (a) Eliminação de $\to$: $\neg(\neg(F(a) \land \forall x[\neg F(x) \lor G(x)]) \lor G(a))$
   (b) Redução do domínio de $\neg$: $\neg\neg(F(a) \land \forall x[\neg F(x) \lor G(x)]) \land \neg G(a)$
   $F(a) \land \forall x[\neg F(x) \lor G(x)] \land \neg G(a)$
   (c) Normalização de variáveis: já está.
   (d) Eliminação de $\exists$: já está.
   (e) Mover $\forall$ para esquerda: $\forall x[F(a) \land (\neg F(x) \lor G(x)) \land \neg G(a)]$
   (f) Eliminação de $\forall$: $F(a) \land (\neg F(x) \lor G(x)) \land \neg G(a)$
   (g) Obtenção da forma conjuntiva normal: já está.
   (h) Eliminação de $\land$: $\{F(a), \neg F(x) \lor G(x), \neg G(a)\}$
   (i) Eliminação de $\lor$: $\{\{F(a)\}, \{\neg F(x), G(x)\}, \{\neg G(a)\}\}$

   Aplicação do princípio da resolução:



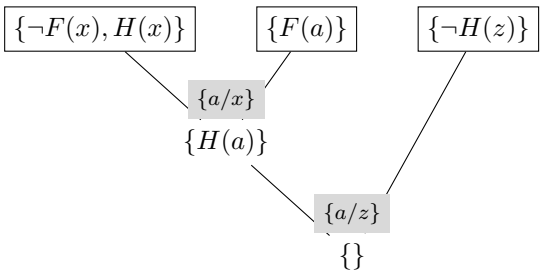
2. $(\forall x[F(x) \to G(x)] \land \forall x[G(x) \to H(x)]) \to \forall x[F(x) \to H(x)]$
   Passagem para a forma clausal de
   $\neg((\forall x[F(x) \to G(x)] \land \forall x[G(x) \to H(x)]) \to \forall x[F(x) \to H(x)])$:
   (a) Eliminação de $\to$: $\neg(\neg(\forall x[\neg F(x) \lor G(x)] \land \forall x[\neg G(x) \lor H(x)]) \lor \forall x[\neg F(x) \lor H(x)])$
   (b) Redução do domínio de $\neg$: $\neg\neg(\forall x[\neg F(x) \lor G(x)] \land \forall x[\neg G(x) \lor H(x)]) \land \neg\forall x[\neg F(x) \lor H(x)]$
   $(\forall x[\neg F(x) \lor G(x)] \land \forall x[\neg G(x) \lor H(x)]) \land \exists x[F(x) \land \neg H(x)]$
   (c) Normalização de variáveis: $(\forall x[\neg F(x) \lor G(x)] \land \forall y[\neg G(y) \lor H(y)]) \land \exists z[F(z) \land \neg H(z)]$
   (d) Eliminação de $\exists$: $(\forall x[\neg F(x) \lor G(x)] \land \forall y[\neg G(y) \lor H(y)]) \land (F(a) \land \neg H(a))$
   (e) Mover $\forall$ para esquerda: $\forall x, y[(\neg F(x) \lor G(x)) \land (\neg G(y) \lor H(y)) \land (F(a) \land \neg H(a))]$
   (f) Eliminação de $\forall$: $(\neg F(x) \lor G(x)) \land (\neg G(y) \lor H(y)) \land (F(a) \land \neg H(a))$
   (g) Obtenção da forma conjuntiva normal: já está.
   (h) Eliminação de $\land$: $\{\neg F(x) \lor G(x), \neg G(y) \lor H(y), F(a), \neg H(a)\}$
   (i) Eliminação de $\lor$: $\{\{\neg F(x), G(x)\}, \{\neg G(y), H(y)\}, \{F(a)\}, \{\neg H(a)\}\}$

   Aplicação do princípio da resolução:



3. $(\forall x[F(x) \rightarrow H(x)] \wedge \exists y[F(y)]) \rightarrow \exists z[H(z)]$
   Passagem para a forma clausal de $\neg((\forall x[F(x) \rightarrow H(x)] \wedge \exists y[F(y)]) \rightarrow \exists z[H(z)])$:
   (a) Eliminação de $\rightarrow$: $\neg(\neg(\forall x[\neg F(x) \vee H(x)] \wedge \exists y[F(y)]) \vee \exists z[H(z)])$
   (b) Redução do domínio de $\neg$: $\neg\neg(\forall x[\neg F(x) \vee H(x)] \wedge \exists y[F(y)]) \wedge \neg\exists z[H(z)])$
   $\forall x[\neg F(x) \vee H(x)] \wedge \exists y[F(y)] \wedge \forall z[\neg H(z)])$
   (c) Normalização de variáveis: já está.
   (d) Eliminação de $\exists$: $\forall x[\neg F(x) \vee H(x)] \wedge F(a) \wedge \forall z[\neg H(z)])$
   (e) Mover $\forall$ para esquerda: $\forall x, z[(\neg F(x) \vee H(x)) \wedge F(a) \wedge \neg H(z)]$
   (f) Eliminação de $\forall$: $(\neg F(x) \vee H(x)) \wedge F(a) \wedge \neg H(z)$
   (g) Obtenção da forma conjuntiva normal: já está.
   (h) Eliminação de $\wedge$: $\{\neg F(x) \vee H(x), F(a), \neg H(z)\}$
   (i) Eliminação de $\vee$: $\{\{\neg F(x), H(x)\}, \{F(a)\}, \{\neg H(z)\}\}$

   Aplicação do princípio da resolução:

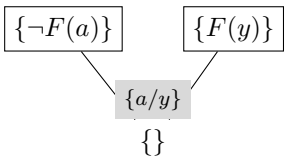


Exercício 11.7
Demonstre os teoremas do exercício 8.3, que correspondem a aplicações das leis de De Morgan para os quantificadores, usando resolução.

Resposta:

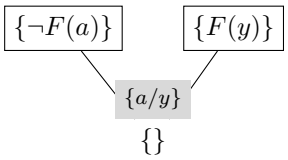
1. $\neg\forall x[F(x)] \rightarrow \exists x[\neg F(x)]$
   Passagem para a forma clausal de $\neg(\neg\forall x[F(x)] \rightarrow \exists x[\neg F(x)])$:
   (a) Eliminação de $\rightarrow$: $\neg(\neg\neg\forall x[F(x)] \vee \exists x[\neg F(x)])$
   (b) Redução do domínio de $\neg$: $\neg\forall x[F(x)] \wedge \neg\exists x[\neg F(x)]$
   $\exists x[\neg F(x)] \wedge \forall x[F(x)]$
   (c) Normalização de variáveis: $\exists x[\neg F(x)] \wedge \forall y[F(y)]$
   (d) Eliminação de $\exists$: $\neg F(a) \wedge \forall y[F(y)]$
   (e) Mover $\forall$ para esquerda: $\forall y[\neg F(a) \wedge F(y)]$
   (f) Eliminação de $\forall$: $\neg F(a) \wedge F(y)$
   (g) Obtenção da forma conjuntiva normal: já está.
   (h) Eliminação de $\wedge$: $\{\neg F(a), F(y)\}$
   (i) Eliminação de $\vee$: $\{\{\neg F(a)\}, \{F(y)\}\}$

   Aplicação do princípio da resolução:



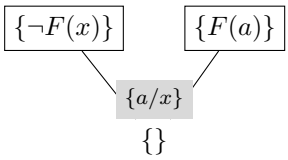
2. $\exists x[\neg F(x)] \rightarrow \neg\forall x[F(x)]$
   Passagem para a forma clausal de $\neg(\exists x[\neg F(x)] \rightarrow \neg\forall x[F(x)])$:
   (a) Eliminação de $\rightarrow$: $\neg(\neg\exists x[\neg F(x)] \vee \neg\forall x[F(x)])$
   (b) Redução do domínio de $\neg$: $\neg\neg\exists x[\neg F(x)] \wedge \neg\neg\forall x[F(x)]$
   $\exists x[\neg F(x)] \wedge \forall x[F(x)]$
   (c) Normalização de variáveis: $\exists x[\neg F(x)] \wedge \forall y[F(y)]$
   (d) Eliminação de $\exists$: $\neg F(a) \wedge \forall y[F(y)]$
   (e) Mover $\forall$ para esquerda: $\forall y[\neg F(a) \wedge F(y)]$
   (f) Eliminação de $\forall$: $\neg F(a) \wedge F(y)$
   (g) Obtenção da forma conjuntiva normal: já está.
   (h) Eliminação de $\wedge$: $\{\neg F(a), F(y)\}$
   (i) Eliminação de $\vee$: $\{\{\neg F(a)\}, \{F(y)\}\}$

   Aplicação do princípio da resolução:



3. $\neg\exists x[F(x)] \rightarrow \forall x[\neg F(x)]$
   Passagem para a forma clausal de $\neg(\neg\exists x[F(x)] \rightarrow \forall x[\neg F(x)])$:
   (a) Eliminação de $\rightarrow$: $\neg(\neg\neg\exists x[F(x)] \vee \forall x[\neg F(x)])$
   (b) Redução do domínio de $\neg$: $\neg\exists x[F(x)] \wedge \neg\forall x[\neg F(x)]$
   $\forall x[\neg F(x)] \wedge \exists x[F(x)]$
   (c) Normalização de variáveis: $\forall x[\neg F(x)] \wedge \exists y[F(y)]$
   (d) Eliminação de $\exists$: $\forall x[\neg F(x)] \wedge F(a)$
   (e) Mover $\forall$ para esquerda: $\forall x[\neg F(x) \wedge F(a)]$
   (f) Eliminação de $\forall$: $\neg F(x) \wedge F(a)$
   (g) Obtenção da forma conjuntiva normal: já está.
   (h) Eliminação de $\wedge$: $\{\neg F(x), F(a)\}$
   (i) Eliminação de $\vee$: $\{\{\neg F(x)\}, \{F(a)\}\}$

   Aplicação do princípio da resolução:



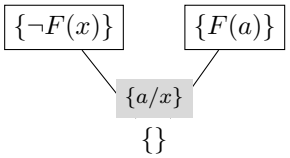
4. $\forall x[\neg F(x)] \rightarrow \neg\exists x[F(x)]$
   Passagem para a forma clausal de $\neg(\forall x[\neg F(x)] \rightarrow \neg\exists x[F(x)])$:
   (a) Eliminação de $\rightarrow$: $\neg(\neg\forall x[\neg F(x)] \vee \neg\exists x[F(x)])$
   (b) Redução do domínio de $\neg$: $\neg\neg\forall x[\neg F(x)] \wedge \neg\neg\exists x[F(x)]$
   $\forall x[\neg F(x)] \wedge \exists x[F(x)]$
   (c) Normalização de variáveis: $\forall x[\neg F(x)] \wedge \exists y[F(y)]$
   (d) Eliminação de $\exists$: $\forall x[\neg F(x)] \wedge F(a)$
   (e) Mover $\forall$ para esquerda: $\forall x[\neg F(x) \wedge F(a)]$
   (f) Eliminação de $\forall$: $\neg F(x) \wedge F(a)$

(g) Obtenção da forma conjuntiva normal: já está.

(h) Eliminação de $\wedge$: $\{\neg F(x), F(a)\}$

(i) Eliminação de $\vee$: $\{\{\neg F(x)\}, \{F(a)\}\}$

Aplicação do princípio da resolução:



Exercício 11.8
Considere as seguintes afirmações:

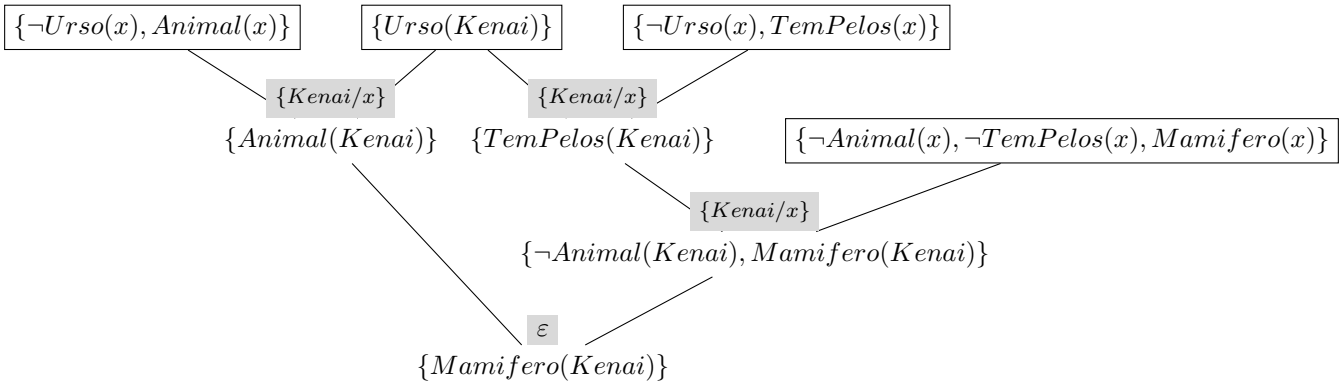
- Os animais com pêlos são mamíferos.
- Os ursos são animais com pêlos.
- Os coelhos são mamíferos.
- O Kenai é um urso.
- O Bugsbunny é um coelho.
- O Sylvester é um animal com pêlos.

1. Represente-as em lógica de primeira ordem.
2. Usando resolução, responda às seguintes perguntas:
   (a) O Kenai é mamífero?
   (b) Quem é que tem pêlos?
   (c) Quais são os mamíferos?

Resposta:

1. Em lógica de primeira ordem, ficam:
   - $\forall x[(Animal(x) \wedge TemPelos(x)) \rightarrow Mamifero(x)]$
   - $\forall x[Urso(x) \rightarrow (Animal(x) \wedge TemPelos(x))]$
   - $\forall x[Coelho(x) \rightarrow Mamifero(x)]$
   - $Urso(Kenai)$
   - $Coelho(Bugsbunny)$
   - $Animal(Sylvester) \wedge TemPelos(Sylvester)$
2. Para responder às perguntas, é necessário passar cada uma das fórmulas correspondentes às premissas do problema para a forma clausal.
   - $\{\neg Animal(x), \neg TemPelos(x), Mamifero(x)\}$
   - $\{\neg Urso(x) \vee (Animal(x) \wedge TemPelos(x))\}$
     $\{\neg Urso(x), Animal(x)\}, \{\neg Urso(x), TemPelos(x)\}$
   - $\{\neg Coelho(x), Mamifero(x)\}$
   - $\{Urso(Kenai)\}$
   - $\{Coelho(Bugsbunny)\}$
   - $\{Animal(Sylvester)\}, \{TemPelos(Sylvester)\}$

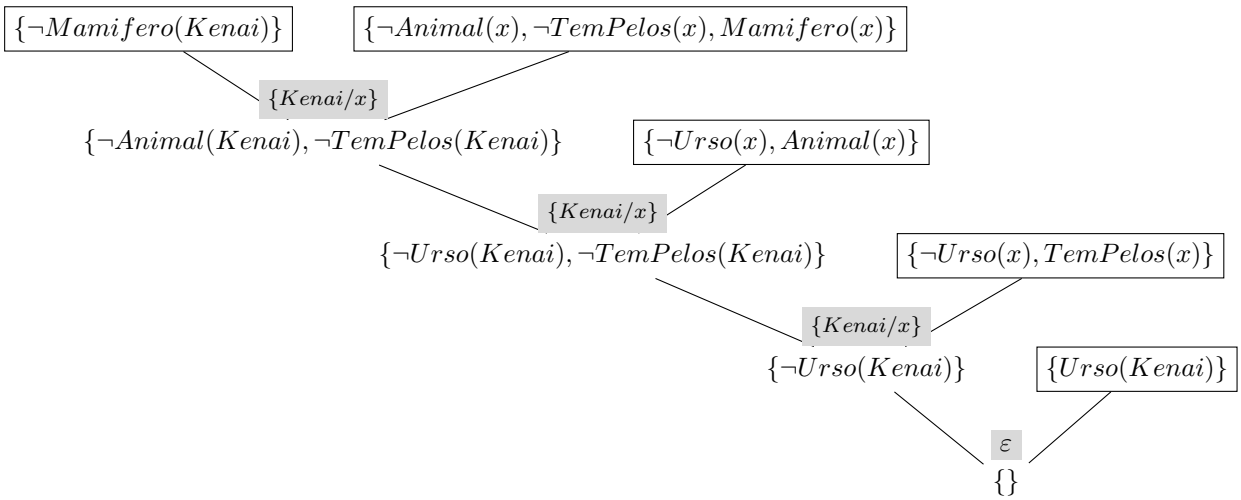
(a) Para saber se o Kenai é mamífero, é necessário responder a uma questão do tipo verdadeiro ou falso. Neste caso, pretendemos saber se a cláusula $Mamifero(Kenai)$ pode ser derivada a partir do conjunto de cláusulas inicial. Uma vez que nem todas as cláusulas iniciais (ou premissas) são necessárias, vamos representar apenas as que são usadas na prova.



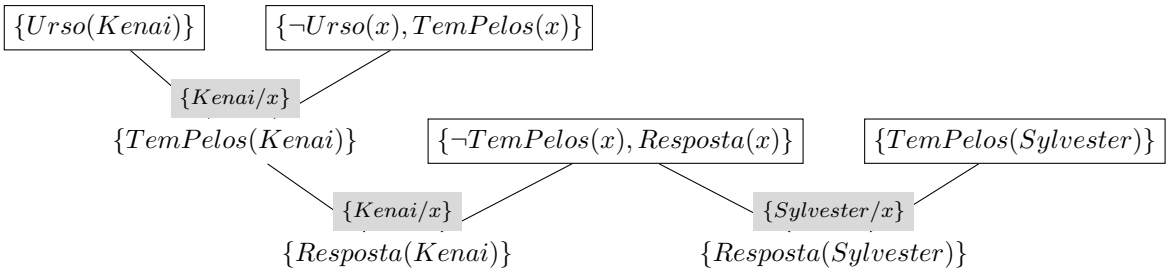
Também podemos usar resolução no formato de prova:

1	$\{\neg Urso(x), Animal(x)\}$	Prem
2	$\{Urso(Kenai)\}$	Prem
3	$\{\neg Urso(x), TemPelos(x)\}$	Prem
4	$\{\neg Animal(x), \neg TemPelos(x), Mamifero(x)\}$	Prem
5	$\{Animal(Kenai)\}$	Res, (1, 2), $\{Kenai/x\}$
6	$\{TemPelos(Kenai)\}$	Res, (2, 3), $\{Kenai/x\}$
7	$\{\neg Animal(Kenai), Mamifero(Kenai)\}$	Res, (4, 6), $\{Kenai/x\}$
8	$\{Mamifero(Kenai)\}$	Res, (5, 7), $\{\}$

Na realidade, e uma vez que estamos a usar resolução, o mais normal é negar o que se quer provar e tentar chegar a uma contradição. Assim, a maneira mais comum de responder a esta questão usando resolução seria através da seguinte prova por redução ao absurdo:



(b) Para saber quem é que tem pêlos, estamos interessados em saber quais são as instâncias que fazem com que a fbf $TemPelos(x)$ seja consequência do conjunto de premissas. Neste caso, para além de necessitarmos de transformar as premissas para a forma clausal, teremos que adicionar a fbf $\forall x[TemPelos(x) \rightarrow Resposta(x)]$ (ou, na forma clausal, $\{\neg TemPelos(x), Resposta(x)\}$), que especifica quais são as respostas desejadas.

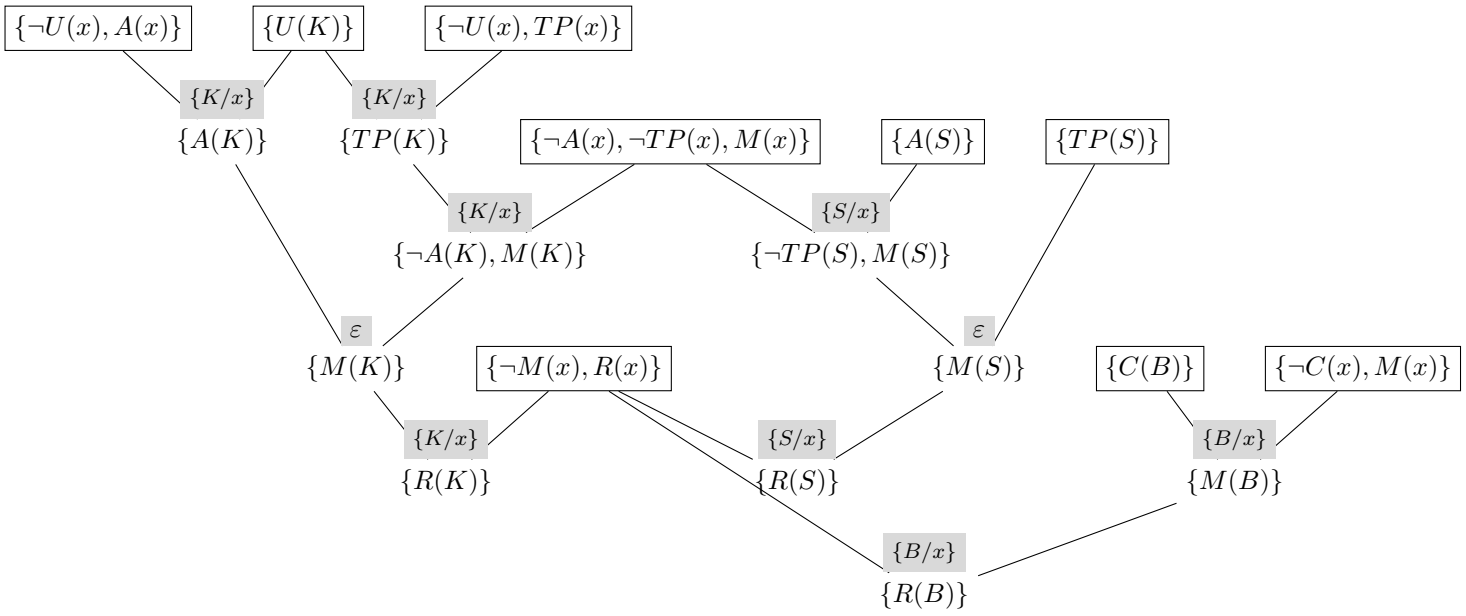


Também podemos usar resolução no formato de prova:

1	$\{Urso(Kenai)\}$	Prem
2	$\{\neg Urso(x), TemPelos(x)\}$	Prem
3	$\{TemPelos(Sylvester)\}$	Prem
4	$\{\neg TemPelos(x), Resposta(x)\}$	Prem
5	$\{TemPelos(Kenai)\}$	Res, (1, 2), $\{Kenai/x\}$
6	$\{Resposta(Kenai)\}$	Res, (4, 5), $\{Kenai/x\}$
7	$\{Resposta(Sylvester)\}$	Res, (3, 4), $\{Sylvester/x\}$

Convém notar que não conseguimos saber se o Bugsbunny tem pêlos. Ele é um coelho e os coelhos são mamíferos, mas o que nós sabemos é que os animais com pêlos são mamíferos e não que os mamíferos são animais com pêlos. Na realidade, deveríamos ter representado esta implicação como uma equivalência, se não fossem as excepções dos golfinhos e baleias...

(c) Para saber quais são os mamíferos, estamos interessados em saber quais são as instâncias que fazem com que a fbf $Mamifero(x)$ seja consequência do conjunto de premissas. Neste caso, adicionamos a fbf $\forall x[Mamifero(x) \rightarrow Resposta(x)]$ (ou, na forma clausal, $\{\neg Mamifero(x), Resposta(x)\}$). Nesta árvore, abreviamos cada símbolo para a primeira letra de cada palavra que o compõe.



12 Programação em Lógica — Resolução SLD; Árvores SLD

Sumário:

- Cláusulas de Horn
- Programas, objectivos, função de selecção
- Resolução SLD
- Árvores SLD

Resumo:

Uma *cláusula* é uma disjunção de literais.

Uma *cláusula de Horn* é uma cláusula que contém, no máximo, um literal positivo. Numa cláusula de Horn, o literal positivo (se existir) é chamado a *cabeça* da cláusula e os literais negativos (se existirem) são chamados o *corpo* da cláusula.

A fbf $(A \wedge B) \rightarrow C$ é transformada na cláusula $\{\neg A, \neg B, C\}$, que só tem um literal positivo e por isso pode ser transformada na cláusula de Horn $C \leftarrow A, B$, ou seja, a implicação da fbf inicial é transformada na "implicação ao contrário" da cláusula de Horn. Esta cláusula de Horn significa: para provar C, provar A e provar B.

Existem quatro tipos de cláusulas de Horn:

1. *Regras* ou *implicações*, correspondendo a cláusulas em que tanto a cabeça como o corpo contêm literais, por exemplo, $C \leftarrow P_1, P_2$.
2. *Afirmações* ou *factos*, correspondendo a cláusulas em que o corpo não contém literais (ou seja, é vazio) mas a cabeça contém um literal, por exemplo, $C \leftarrow$.
3. *Objectivos*, correspondendo a cláusulas em que a cabeça não contém um literal (ou seja, é vazia) mas o corpo contém pelo menos um literal, por exemplo, $\leftarrow P_1, P_2$.
4. A *cláusula vazia*, ou seja, $\square$.

As cláusulas do tipo 1 e 2 (ou seja, as regras e as afirmações, que têm cabeça) têm o nome de *cláusulas determinadas* (do Inglês "definite clauses").

Regra da resolução: assumindo que o literal α unifica com o literal β_i, sendo s o unificador mais geral destes dois literais, o caso geral da aplicação da regra da resolução a cláusulas de Horn é o seguinte:

$$\begin{array}{lll} m & \alpha \leftarrow \gamma_1, \ldots, \gamma_m & \\ n & \delta \leftarrow \beta_1, \ldots, \beta_{i-1}, \beta_i, \beta_{i+1}, \ldots, \beta_n & \\ n+1 & (\delta \leftarrow \beta_1, \ldots, \beta_{i-1}, \gamma_1, \ldots, \gamma_m, \beta_{i+1}, \ldots, \beta_n) \cdot s & \text{Res, } (m, n) \end{array}$$

Em programação em lógica, um *programa* é qualquer conjunto finito de cláusulas determinadas e um *objectivo* corresponde a uma cláusula que se pretende derivar a partir desse programa. Num programa, o conjunto de todas as cláusulas cuja cabeça corresponde a

um literal contendo a letra de predicado P diz-se a *definição* de P. Uma definição de um predicado que contenha apenas cláusulas fechadas chama-se uma *base de dados*.

Resposta de um programa a um objectivo: Sendo Δ um programa e α um objectivo, uma resposta de Δ a α é uma substituição s para as variáveis de α. Uma resposta s de Δ a α diz-se correcta se $\Delta \models (\alpha \cdot s)$.

Uma *função de selecção*, S, é uma regra para escolher um literal numa cláusula objectivo como candidato à aplicação do princípio da resolução. Esta é uma função do conjunto dos objectivos para o conjunto dos literais, tal que $S(\leftarrow \alpha_1, \ldots, \alpha_n) \in \{\alpha_1, \ldots, \alpha_n\}$.

A *resolução SLD* (Linear resolution with Selection function and Definite clauses) é uma estratégia de resolução linear aplicável a cláusulas determinadas, conjuntamente com uma função de selecção, a qual, dentro dos possíveis literais aplicáveis com a regra da resolução, escolhe um literal de um modo determinístico.

Uma *regra de procura* é uma função do conjunto dos literais e do conjunto dos programas para o conjunto das cláusulas definidas, tal que $P(\alpha, \Delta) \in \Delta$.

Árvore SLD: Seja Δ um programa, α um objectivo e S uma função de selecção. A árvore SLD de Δ via S é uma árvore rotulada, construída do seguinte modo:

1. O rótulo de cada nó é um objectivo;
2. O rótulo da raiz é α;
3. Cada nó com rótulo $\leftarrow \beta_1, \ldots, \beta_n$ tem um ramo por cada cláusula $\delta \leftarrow \gamma_1, \ldots, \gamma_n$ de Δ cuja cabeça seja unificável com $S(\leftarrow \beta_1, \ldots, \beta_n)$. O rótulo da raiz deste ramo corresponde ao resolvente entre as duas cláusulas.

Um ramo cuja folha tem o rótulo $\square$ diz-se um nó *bem sucedido*; um ramo cuja folha tem um rótulo que não corresponda à cláusula vazia diz-se um nó *falhado*; os restantes ramos dizem-se ramos *infinitos*.

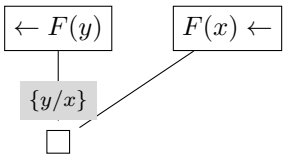
Seja Δ um programa e α um objectivo. Então, independentemente da função de selecção, todas as árvores SLD de Δ e α têm o mesmo número (finito ou infinito) de ramos bem sucedidos. Em termos computacionais, este resultado afirma que a qualidade de uma implementação da resolução SLD é independente da função de selecção escolhida, pelo menos no que diz respeito às respostas que podem ser calculadas.

Exercício 12.1
Demonstre os argumentos do exercício 8.1, usando resolução SLD e uma função de selecção que escolhe o primeiro literal da cláusula objectivo.
Notas: Como está a tentar demonstrar argumentos, pode fazer provas por refutação em que algumas das cláusulas correspondem à negação da conclusão do argumento. Uma vez que a passagem para a forma clausal já foi feita na aula sobre resolução, apresentam-se com cada argumento as cláusulas que lhe correspondem.

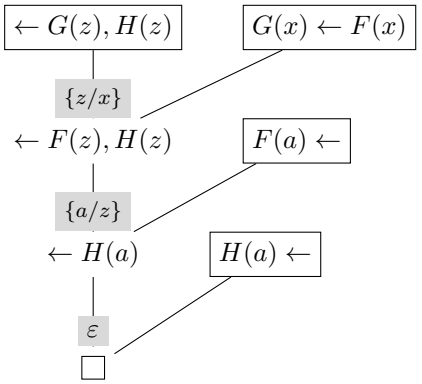
1. Argumento: $(\{\forall x[F(x)]\}, \exists x[F(x)])$
   Cláusulas: $\{\{F(x)\}, \{\neg F(y)\}\}$
2. Argumento: $(\{\forall x[F(x) \rightarrow G(x)], \exists x[F(x) \wedge H(x)]\}, \exists x[G(x) \wedge H(x)])$
   Cláusulas: $\{\{\neg F(x), G(x)\}, \{F(a)\}, \{H(a)\}, \{\neg G(z), \neg H(z)\}\}$

Resposta:

1. Cláusulas: $\{\{F(x)\}, \{\neg F(y)\}\}$
   Cláusulas de Horn: $F(x) \leftarrow$
   $\leftarrow F(y)$
   Aplicação da resolução SLD com o objectivo $\leftarrow F(y)$:



   A resposta é $(\{y/x\})|_{\{y\}} = \varepsilon$
   Nota: a resposta é sempre uma substituição para as variáveis do objectivo.
2. Cláusulas: $\{\{\neg F(x), G(x)\}, \{F(a)\}, \{H(a)\}, \{\neg G(z), \neg H(z)\}\}$
   Cláusulas de Horn: $G(x) \leftarrow F(x)$
   $F(a) \leftarrow$
   $H(a) \leftarrow$
   $\leftarrow G(z), H(z)$
   Aplicação da resolução SLD com o objectivo $\leftarrow G(z), H(z)$:



   A resposta é $(\{z/x\} \circ \{a/z\} \circ \varepsilon)|_{\{z\}} = \{a/z\}$

Exercício 12.2
Demonstre os teoremas do exercício 8.2, usando resolução SLD e uma função de selecção que escolhe o último literal da cláusula objectivo.

Notas: Como está a tentar provar se uma fórmula é um teorema, deve fazer provas por refutação em que as cláusulas correspondem à negação da fórmula inicial. Uma vez que a passagem para a forma clausal já foi feita na aula sobre resolução, apresentam-se com cada fórmula as cláusulas que correspondem à sua negação.

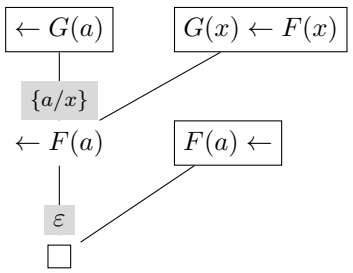
1. $(F(a) \land \forall x[F(x) \to G(x)]) \to G(a)$
   $\{\{F(a)\}, \{\neg F(x), G(x)\}, \{\neg G(a)\}\}$

2. $(\forall x[F(x) \to G(x)] \land \forall y[G(y) \to H(y)]) \to \forall z[F(z) \to H(z)]$
   $\{\{\neg F(x), G(x)\}, \{\neg G(y), H(y)\}, \{F(a)\}, \{\neg H(a)\}\}$

3. $(\forall x[F(x) \to H(x)] \land \exists y[F(y)]) \to \exists z[H(z)]$
   $\{\{\neg F(x), H(x)\}, \{F(a)\}, \{\neg H(z)\}\}$

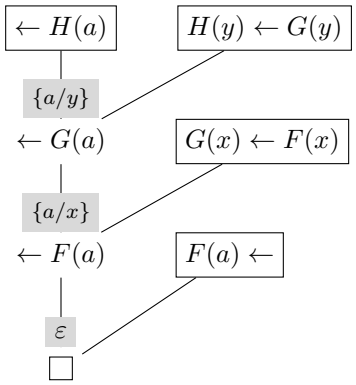
Resposta:

1. Cláusulas: $\{\{F(a)\}, \{\neg F(x), G(x)\}, \{\neg G(a)\}\}$
   Cláusulas de Horn: $F(a) \leftarrow$
   $G(x) \leftarrow F(x)$
   $\leftarrow G(a)$
   Aplicação da resolução SLD com o objectivo $\leftarrow G(a)$:



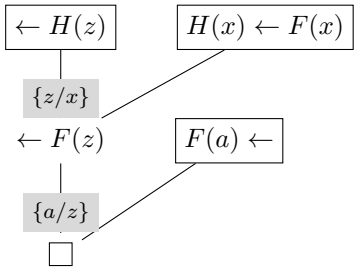
   A resposta é $(\{a/x\} \circ \varepsilon)|_{\{\}} = \varepsilon$

2. Cláusulas: $\{\{\neg F(x), G(x)\}, \{\neg G(y), H(y)\}, \{F(a)\}, \{\neg H(a)\}\}$
   Cláusulas de Horn: $G(x) \leftarrow F(x)$
   $H(y) \leftarrow G(y)$
   $F(a) \leftarrow$
   $\leftarrow H(a)$
   Aplicação da resolução SLD com o objectivo $\leftarrow H(a)$:



   A resposta é $(\{a/y\} \circ \{a/x\} \circ \varepsilon)|_{\{\}} = \varepsilon$

3. Cláusulas: $\{\{\neg F(x), H(x)\}, \{F(a)\}, \{\neg H(z)\}\}$
   Cláusulas de Horn: $H(x) \leftarrow F(x)$
   $F(a) \leftarrow$
   $\leftarrow H(z)$
   Aplicação da resolução SLD com o objectivo $\leftarrow H(z)$:



   A resposta é $(\{z/x\} \circ \{a/z\})|_{\{z\}} = \{a/z\}$

Exercício 12.3
Considere o seguinte conjunto de cláusulas:

- $\{\neg Animal(x), \neg TemPelos(x), Mamifero(x)\}$
- $\{\neg Urso(x), Animal(x)\}$
- $\{\neg Urso(x), TemPelos(x)\}$
- $\{\neg Coelho(x), Mamifero(x)\}$
- $\{Urso(Kenai)\}$
- $\{Coelho(Bugsbunny)\}$
- $\{Animal(Sylvester)\}$
- $\{TemPelos(Sylvester)\}$

Usando resolução SLD e uma função de selecção à sua escolha, responda às seguintes perguntas:

1. O Kenai é mamífero?
2. Quem é que tem pêlos?
3. Quais são os mamíferos?

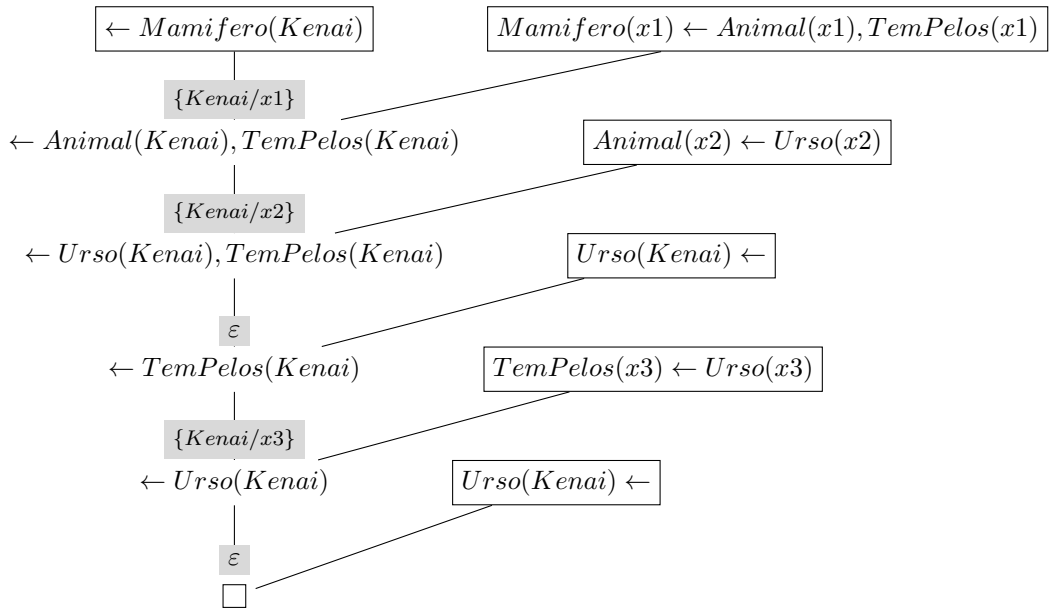
Resposta:

Para usar resolução SLD, é preciso transformar as cláusulas dadas no enunciado em cláusulas de Horn.

$Mamifero(x) \leftarrow Animal(x), TemPelos(x)$
$Animal(x) \leftarrow Urso(x)$
$TemPelos(x) \leftarrow Urso(x)$
$Mamifero(x) \leftarrow Coelho(x)$
$Urso(Kenai) \leftarrow$
$Coelho(Bugsbunny) \leftarrow$
$Animal(Sylvester) \leftarrow$
$TemPelos(Sylvester) \leftarrow$

Vamos usar uma função de selecção que escolhe para unificar o primeiro literal do objectivo.

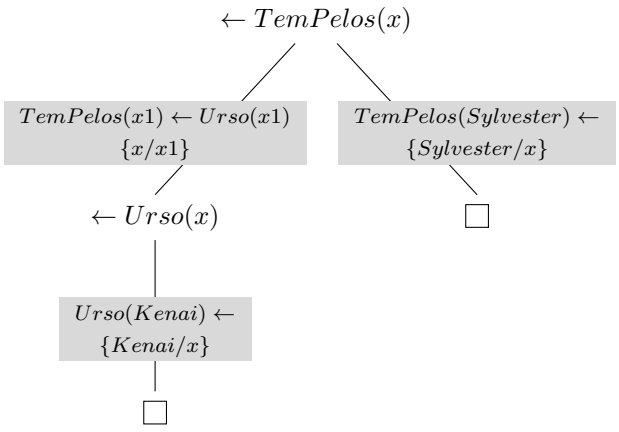
1. Para sabermos se o Kenai é mamífero, vamos aplicar resolução SLD com o objectivo $\leftarrow Mamifero(Kenai)$.



A resposta é $(\{Kenai/x1\} \circ \{Kenai/x2\} \circ \varepsilon \circ \{Kenai/x3\} \circ \varepsilon)|_{\{\}} = \varepsilon$

2. Quem é que tem pêlos?

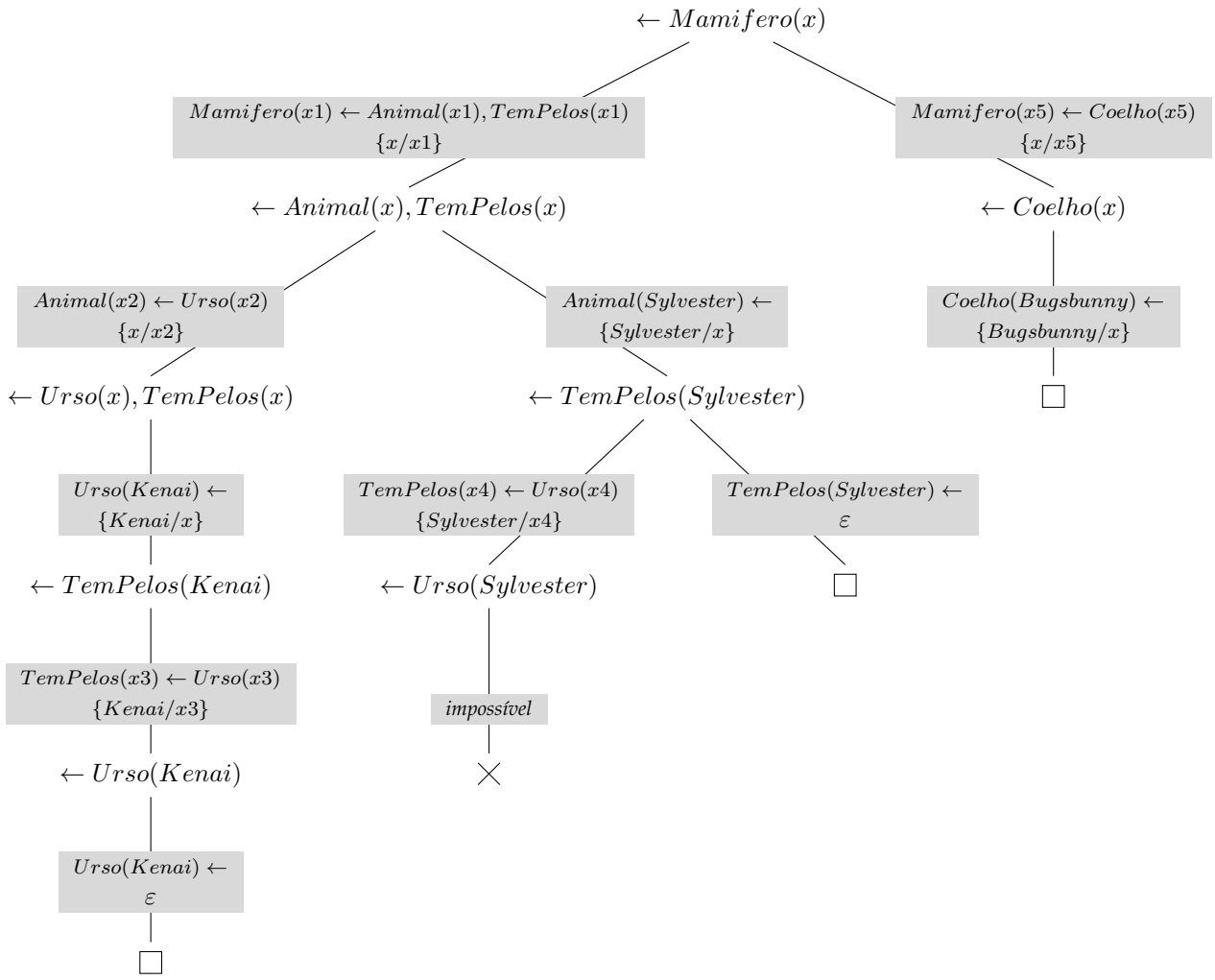
Para saber quem é que tem pêlos, que é uma pergunta que pode ter várias respostas, vamos usar árvores SLD, para garantir que encontramos todas as respostas possíveis.



Temos duas respostas: $\{Kenai/x\}$ e $\{Sylvester/x\}$, ou seja, o $Sylvester$ e o $Kenai$ têm pêlos.

Nota: quando há duas variáveis para unificar, a variável do objectivo substitui a variável do programa que estamos a utilizar, pois isto facilita a composição das substituições para encontrar as respostas; quando existe uma constante e uma variável, a constante substitui sempre a variável, como seria de esperar.

3. Quais são os mamíferos?



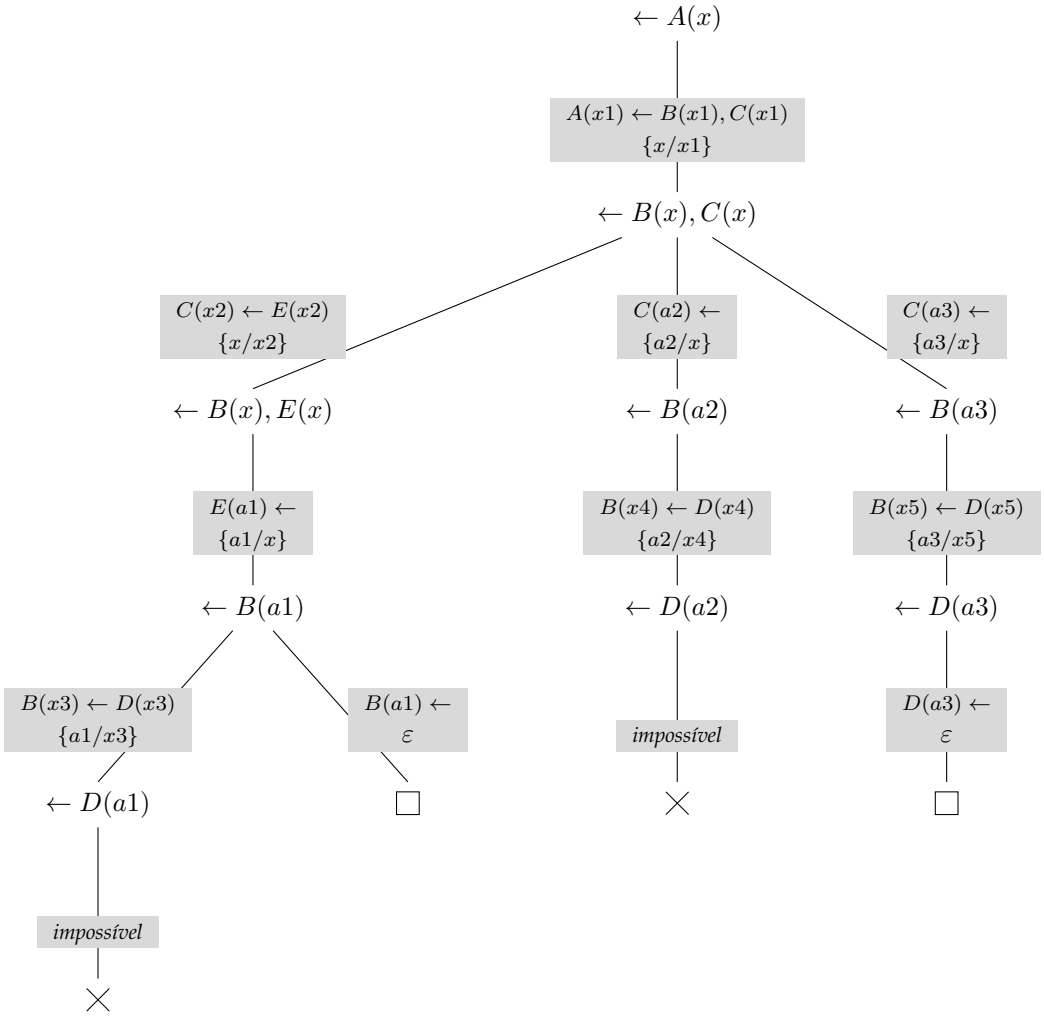
Temos três respostas: $\{Kenai/x\}$, $\{Sylvester/x\}$ e $\{Bugsbunny/x\}$, ou seja, o $Kenai$, o $Sylvester$ e o $Bugsbunny$ são mamíferos.

Exercício 12.4
Considere o seguinte conjunto de cláusulas de Horn:

- $A(x) \leftarrow B(x), C(x)$
- $B(x) \leftarrow D(x)$
- $C(x) \leftarrow E(x)$
- $B(a1) \leftarrow$
- $E(a1) \leftarrow$
- $C(a2) \leftarrow$
- $C(a3) \leftarrow$
- $D(a3) \leftarrow$

Usando uma árvore de resolução SLD e uma função de selecção que escolha para unificar o **último** literal do objectivo, mostre todas as soluções para o seguinte objectivo: $\leftarrow A(x)$. No final, indique explicitamente as soluções.

Resposta:



Este objectivo tem duas soluções: $\{a3/x\}$ e $\{a1/x\}$.

13 Prolog — Árvores de Refutação; Listas

Sumário:

- Prolog
- Árvores de refutação em Prolog
- Listas e operações sobre listas

Resumo:

Um *programa* em Prolog é constituído por uma sequência de frases declarativas escritas em lógica, juntamente com um conjunto de indicações procedimentais que controlam a utilização dessas frases declarativas. Por ser um *conjunto* e não uma *sequência* de cláusulas, a ordem interessa.

- Termos
  - Constantes
    - Átomos — começam com letra minúscula, podem estar entre plicas, átomos especiais.
    - Números — inteiros ou reais.
  - Variáveis — começam com maiúscula ou _. Usam-se variáveis anónimas, representadas por _, quando se quer dizer que existe ali um argumento, mas ele não vai interessar para a computação.
  - Compostos — aplicação de uma função aos seus argumentos.
- Literais — correspondem à aplicação de um predicado ao número apropriado de argumentos. O Prolog tem "overloading" de operadores, isto é, pode haver o mesmo predicado com números de argumentos diferentes e significados diferentes. Ex: `mae(X)` significa que `X` é mãe, `mae(X,Y)` significa que `X` é a mãe de `Y`.
- Cláusulas (de Horn)
  - Afirmações — cláusulas em que o corpo não contém literais.
  - Regras — cláusulas em que tanto a cabeça como o corpo contêm literais.
  - Objectivos — cláusulas em que a cabeça não contém literais mas o corpo contém pelo menos um literal.

O Prolog usa uma estratégia de procura em profundidade com retrocesso e uma função de selecção que escolhe o primeiro literal do objectivo. Isto significa que a ordem das cláusulas e a ordem dos literais no corpo das cláusulas influenciam a ordem pela qual as respostas são encontradas.

O Prolog usa a "closed world assumption", isto é, tudo aquilo que não consegue provar é considerado falso. A resposta do Prolog a um objectivo que não se consegue provar é `No`.

Um objectivo que não contém variáveis pergunta se ele é uma consequência lógica do programa, e a resposta é `Yes` ou `No`. Um objectivo contendo variáveis pergunta se existem alguns valores para as variáveis que façam com que o objectivo seja uma consequência lógica do programa, e a resposta é uma substituição para as variáveis. Escrever `;`

na interacção com o Prolog serve para perguntar se há mais alguma substituição que satisfaça o objectivo.

É possível usar regras lógicas do tipo $(A \vee B) \rightarrow C$ através da utilização de duas cláusulas de Horn `C :- A` e `C :- B`. Fazendo a passagem para a forma clausal,
$(A \vee B) \rightarrow C$
$\neg(A \vee B) \vee C$
$(\neg A \wedge \neg B) \vee C$
$(\neg A \vee C) \wedge (\neg B \vee C)$ — duas cláusulas de Horn.

Não é possível usar regras do tipo $A \rightarrow (B \vee C)$ apenas com cláusulas de Horn. Fazendo a passagem para a forma clausal,
$A \rightarrow (B \vee C)$
$\neg A \vee B \vee C$ — uma cláusula com dois literais positivos não é de Horn.

A redução de um objectivo α por um programa Δ é a substituição de α pelo corpo de uma cláusula de Δ cuja cabeça emparelhe com α.

Listas

O primeiro argumento de uma *lista* tem um elemento e o segundo argumento é recursivamente o resto da lista. A lista vazia é representada por `[]`. O termo `.(X,Y)`, representado por `[X|Y]`, é uma lista com cabeça `X` e resto `Y`. Exemplos de listas: `[]`, `[X]`, `[X,Y]`, `[X|Xs]`, `[X,Y|Ys]`. Definição de listas em Prolog:

```
lista([]).
lista([_|Xs]) :- lista(Xs).
```

Interacção com o Prolog

```
consult('ficheiro').
['ficheiro'].
[user].
;
(.., ..  ; ..)
halt.
```

É possível incluir testes para os predicados no código de um programa, em que se verifica se os resultados são aquilo que se está à espera. Quando o resultado é diferente do indicado, o compilador escreve um aviso a indicar isso. Eu ainda não consegui fazer um exemplo de uma directiva em que a resposta fosse `No`, por exemplo se no exemplo abaixo `X` fosse `4`.

```
/*
  ultimo1(X, Xs) :- tem o valor verdadeiro se X for o último
  elemento da lista Xs.
*/
ultimo1(X, [X]).
ultimo1(X, [_|Xs]) :- ultimo1(X, Xs).

:- ultimo1(X, [1, 2, 3]), X=3.
```

Exercício 13.1
Considere o seguinte programa em Prolog:

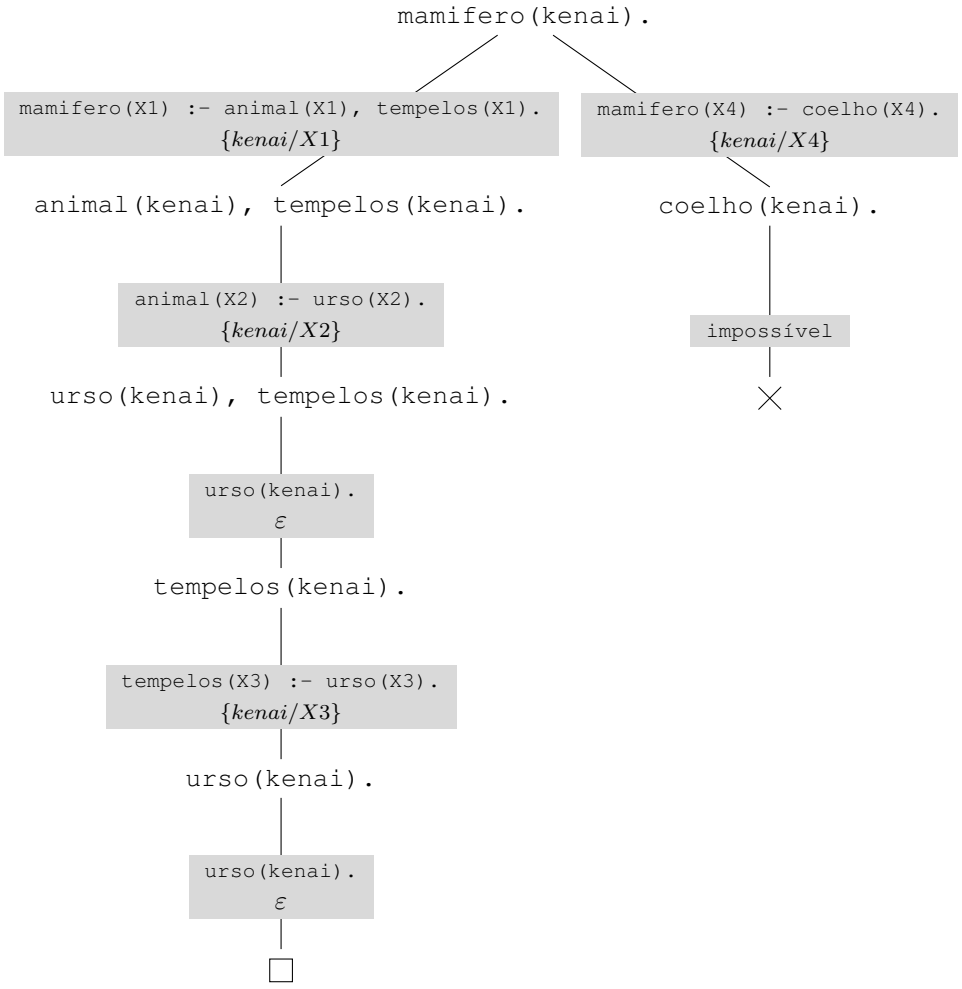
```
mamifero(X) :- animal(X), tempelos(X).
animal(X) :- urso(X).
tempelos(X) :- urso(X).
mamifero(X) :- coelho(X).
urso(kenai).
coelho(bugsbunny).
animal(sylvester).
tempelos(sylvester).
```

Mostrando as árvores de refutação respectivas, explique o que é que o Prolog responderia às seguintes perguntas. Considere que são pedidas todas as soluções em cada um dos casos.

1. O Kenai é mamífero?
2. Quem é que tem pêlos?
3. Quais são os mamíferos?

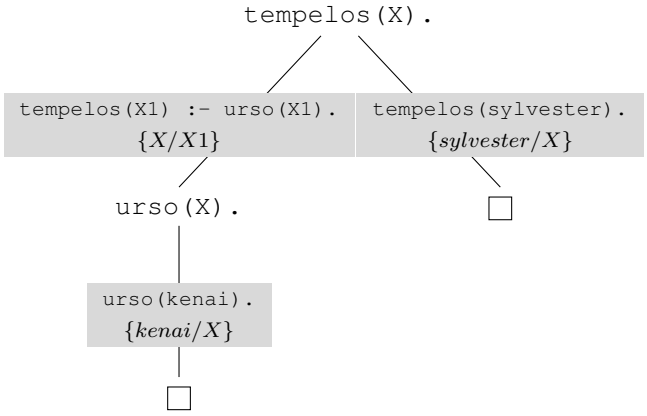
Resposta:

1. Para sabermos se o Kenai é mamífero, usamos o objectivo `mamifero(kenai).`



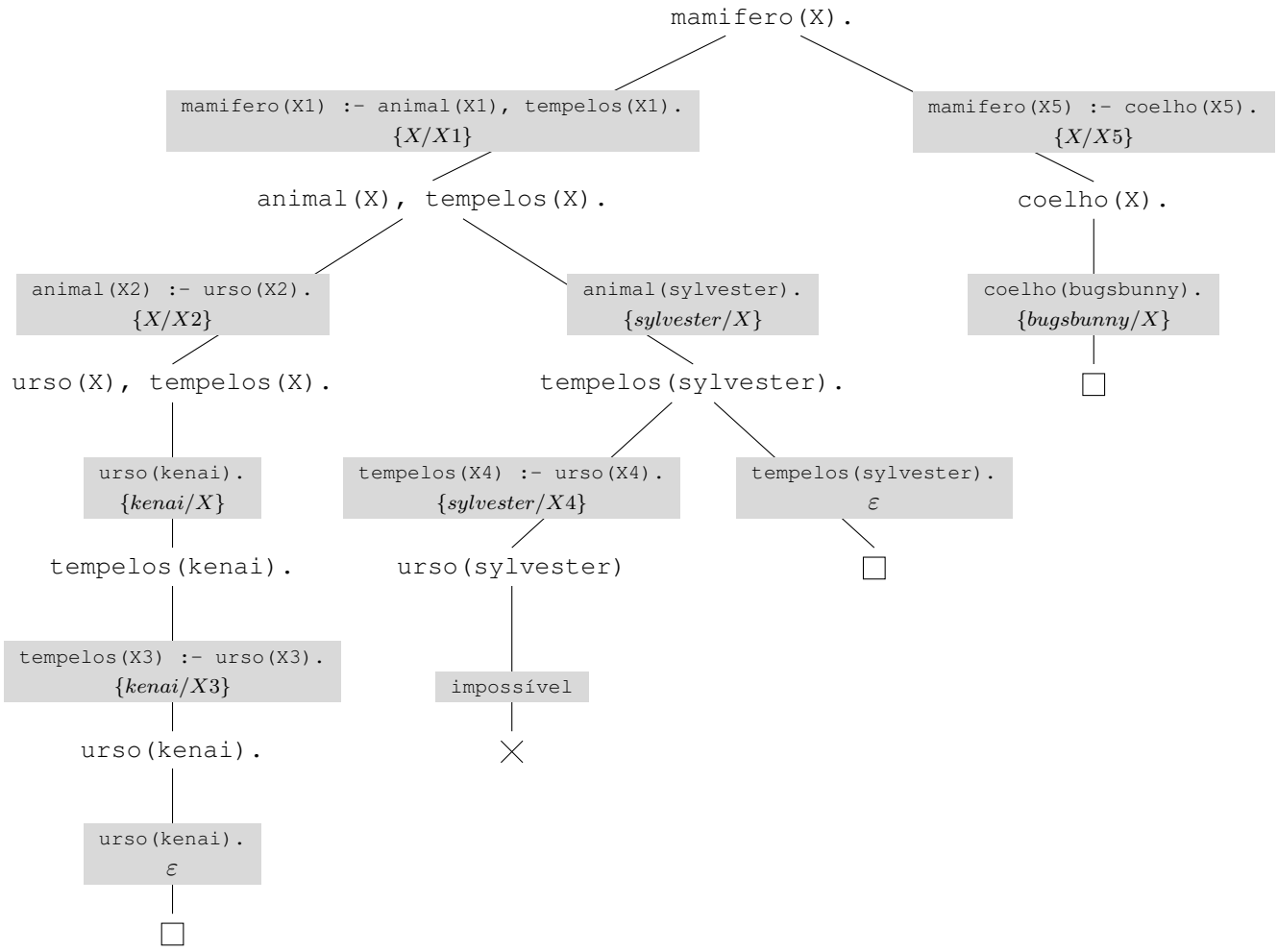
A resposta é `true`.
Como no enunciado diz para considerar que são pedidas todas as soluções em cada um dos casos, o Prolog irá explorar toda a árvore e dar a resposta `true` seguida da resposta `false`.

2. Para saber quem é que tem pêlos, usamos o objectivo `tempelos(X).`



Temos duas respostas: `X = kenai` e `X = sylvester`.

3. Para saber quais são os mamíferos, usamos o objectivo `mamifero(X).`



Temos três respostas: `X = kenai`, `X = sylvester` e `X = bugsbunny`.

Exemplo de uma interacção com o Prolog em que se consulta um ficheiro com este programa e se fazem algumas perguntas.

```
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 6.6.1)
Copyright (c) 1990-2013 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?- consult('~/cadeiras/lpp/13-14/exercicios/mamiferos.pl').
Warning: /Users/acardoso/cadeiras/lpp/13-14/exercicios/mamiferos.pl:4:
Clauses of mamifero/1 are not together in the source-file
Warning: /Users/acardoso/cadeiras/lpp/13-14/exercicios/mamiferos.pl:7:
Clauses of animal/1 are not together in the source-file
Warning: /Users/acardoso/cadeiras/lpp/13-14/exercicios/mamiferos.pl:8:
Clauses of tempelos/1 are not together in the source-file
% /Users/acardoso/cadeiras/lpp/13-14/exercicios/mamiferos.pl compiled 0.00 sec, 9 clau
true.

?- mamifero(kenai).
```

```
true ;
false.

?- tempelos(X).
X = kenai ;
X = sylvester.

?- mamifero(X).
X = kenai ;
X = sylvester ;
X = bugsbunny.

?- animal(X).
X = kenai ;
X = sylvester.

?- animal(bugsbunny).
false.

?- halt.

Process prolog finished
```

Exercício 13.2
Considere definido da seguinte forma o predicado `m(X, Xs)`:

```
m(X, [X|_]).
m(X, [_|Xs]) :- m(X, Xs).
```

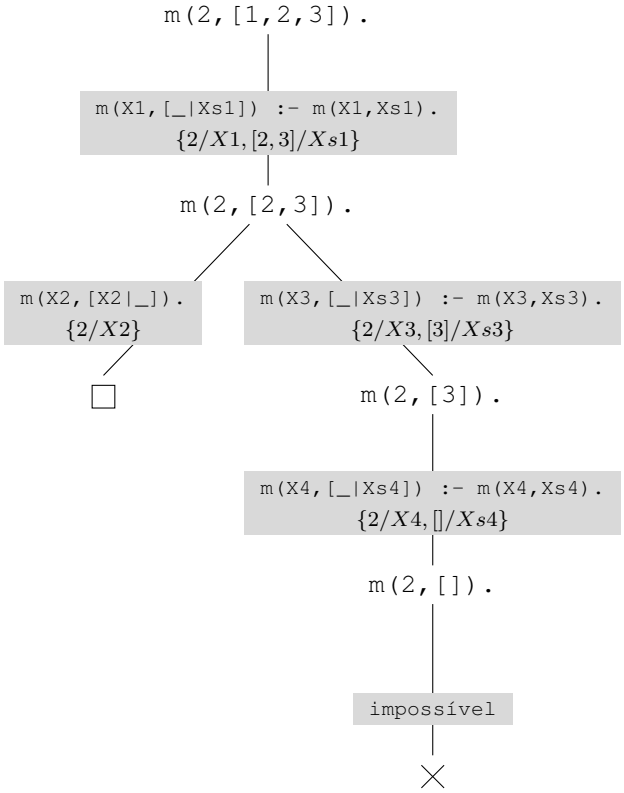
Mostre as árvores de refutação para os seguintes objectivos, considerando que são pedidas todas as soluções.

1. `m(2, [1, 2, 3]).`

2. `m(X, [1, 2, 3]).`

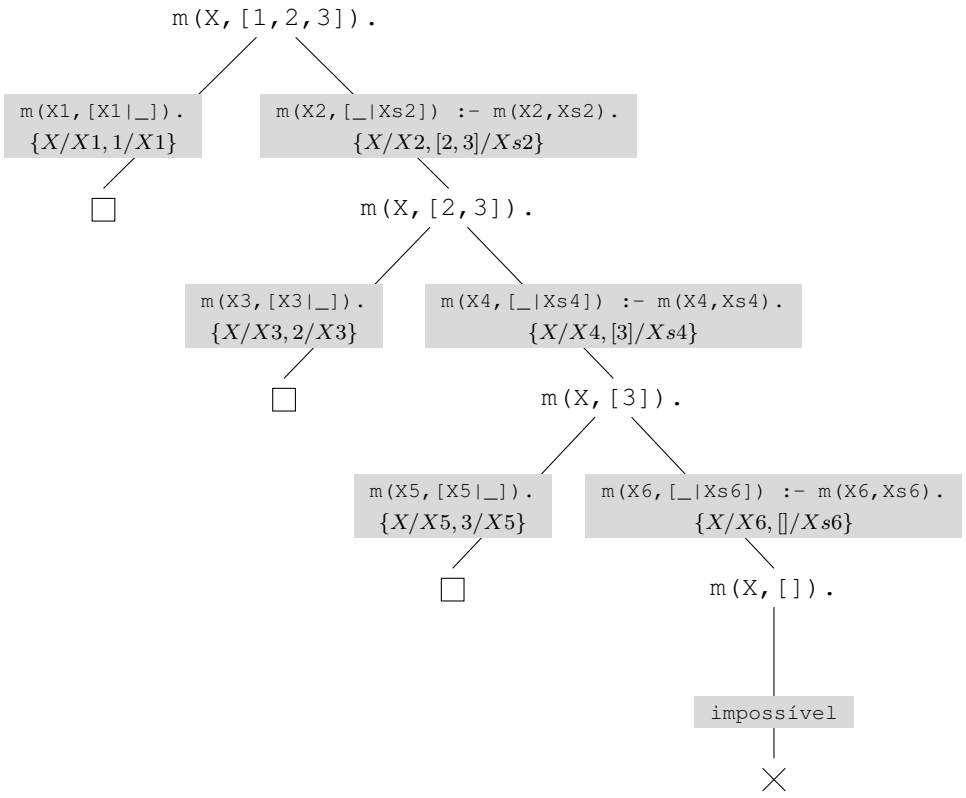
Resposta:

1. `m(2, [1, 2, 3]).`



A resposta é `true`.
Como no enunciado diz para considerar que são pedidas todas as soluções em cada um dos casos, o Prolog irá explorar toda a árvore e dar a resposta `true` seguida da resposta `false`.

2. `m(X, [1, 2, 3]).`



Respostas: `X = 1`; `X = 2`; `X = 3`; e `false`.

Exercício 13.3
(Adaptado de "The Art of Prolog", de Leon Sterling e Ehud Shapiro.)
Defina os seguintes predicados que manipulam listas. Em caso de necessidade, em cada alínea pode usar os predicados definidos nas alíneas anteriores.

1. O predicado `membro(Elemento, Lista)`, que tem o valor verdadeiro se `Elemento` for um membro da lista `Lista`. Por exemplo, `membro(2, [1, 2, 3])` tem o valor verdadeiro.

2. O predicado `prefixo(Prefixo, Lista)`, que tem o valor verdadeiro se `Prefixo` for um prefixo da lista `Lista`. Por exemplo, `prefixo([1, 2], [1, 2, 3])` tem o valor verdadeiro.

3. O predicado `sufixo(Sufixo, Lista)`, que tem o valor verdadeiro se `Sufixo` for um sufixo da lista `Lista`. Por exemplo, `sufixo([2, 3], [1, 2, 3])` tem o valor verdadeiro.

4. O predicado `sublista(Sub, Lista)`, que tem o valor verdadeiro se `Sub` for uma sublista da lista `Lista`. Por exemplo, `sublista([2, 3], [1, 2, 3, 4])` tem o valor verdadeiro.

5. O predicado `junta(Xs, Ys, Zs)`, em que a lista `Zs` é o resultado de concatenar as listas `Xs` e `Ys`. Por exemplo, `junta([1, 2], [3, 4], [1, 2, 3, 4])` tem o valor verdadeiro.

6. Redefina os predicados `membro`, `prefixo`, `sufixo` e `sublista` em termos do predicado `junta`.

7. O predicado `seguidos(X, Y, Zs)`, que tem o valor verdadeiro se `X` e `Y` aparecerem seguidos na lista `Zs`. Por exemplo, `seguidos(2, 3, [1, 2, 3])` tem o valor verdadeiro.

8. O predicado `ultimo(X, Xs)`, que tem o valor verdadeiro se `X` for o último elemento da lista `Xs`. Por exemplo, `ultimo(3, [1, 2, 3])` tem o valor verdadeiro.

9. O predicado `inverte(Xs, Ys)`, que tem o valor verdadeiro se `Ys` for uma lista que contém os elementos de `Xs` pela ordem inversa da qual eles aparecem na lista original. Por exemplo, `inverte([1, 2, 3], [3, 2, 1])` tem o valor verdadeiro.

10. O predicado `comprimento(Xs, N)`, que tem o valor verdadeiro se `N` for o comprimento da lista `Xs`. Por exemplo, `comprimento([a, 1], s(s(0)))` tem o valor verdadeiro.

11. O predicado `repete(Xs, XsXs)`, que tem o valor verdadeiro se cada elemento de `Xs` aparece repetido em `XsXs`. Por exemplo, `repete([1, 2, 3], [1, 1, 2, 2, 3, 3])` tem o valor verdadeiro.

Resposta:

```
/*************************************************************************
  Aula sobre listas
 *************************************************************************/
```

```
/*
  ultimo1(X, Xs) :- tem o valor verdadeiro se X for o ultimo elemento da lista Xs.
*/
ultimo1(X, [X]).
ultimo1(X, [_|Xs]) :- ultimo1(X, Xs).

:- ultimo1(X, [1, 2, 3]), X=3.


/*
  membro(Elemento, Lista) :- tem o valor verdadeiro se Elemento for
  um membro da lista Lista.
*/
membro(X, [X|_]).
membro(X, [_|Ys]) :- membro(X, Ys).

%% ?- membro(1, [1, 2, 3]).
%% true ;
%% false.
%% ?- membro([2], [1, [2], a]).
%% true ;
%% false.
%% ?- membro(a, [1, b, 3]).
%% false.
%% ?- membro(X, [1, 2, 3]).
%% X = 1 ;
%% X = 2 ;
%% X = 3 ;
%% false.
%% ?- membro(1, Y).
%% Y = [1|_G236] ;
%% Y = [_G235, 1|_G239] ;
%% Y = [_G235, _G238, 1|_G242] ;
%% Y = [_G235, _G238, _G241, 1|_G245] ;
%% Y = [_G235, _G238, _G241, _G244, 1|_G248] ;
%% Y = [_G235, _G238, _G241, _G244, _G247, 1|_G251] ;
%% Y = [_G235, _G238, _G241, _G244, _G247, _G250, 1|_G254]
%% ?- membro(X, Y).
%% Y = [X|_G251] ;
%% Y = [_G250, X|_G254] ;
%% Y = [_G250, _G253, X|_G257] ;
%% Y = [_G250, _G253, _G256, X|_G260] ;
%% Y = [_G250, _G253, _G256, _G259, X|_G263] ;
%% Y = [_G250, _G253, _G256, _G259, _G262, X|_G266] ;
%% Y = [_G250, _G253, _G256, _G259, _G262, _G265, X|_G269]


/*
  prefixo(Prefixo, Lista) :- tem o valor verdadeiro se Prefixo for um
  prefixo da lista Lista.
*/
prefixo([], _).
prefixo([X|Xs], [X|Ys]) :- prefixo(Xs, Ys).

%% ?- prefixo([1, 2], [1, 2, 3]).
```

```
%% true.
%% ?- prefixo([1, 2], [2, 2, 3]).
%% false.
%% ?- prefixo([], 5).
%% true.
%% ?- prefixo(X, [1, 2, 3]).
%% X = [] ;
%% X = [1] ;
%% X = [1, 2] ;
%% X = [1, 2, 3] ;
%% false.
%% ?- prefixo([1, 2], Y).
%% Y = [1, 2|_G251].
%% ?- prefixo(X, Y).
%% X = [] ;
%% X = [_G250],
%% Y = [_G250|_G254] ;
%% X = [_G250, _G256],
%% Y = [_G250, _G256|_G260] ;
%% X = [_G250, _G256, _G262],
%% Y = [_G250, _G256, _G262|_G266] ;
%% X = [_G250, _G256, _G262, _G268],
%% Y = [_G250, _G256, _G262, _G268|_G272]


/*
  sufixo(Sufixo, Lista) :- tem o valor verdadeiro se Sufixo for um
  sufixo da lista Lista.
*/
sufixo(Xs, Xs).
sufixo(Xs, [_|Ys]) :- sufixo(Xs, Ys).

%% ?- sufixo([2, 3], [1, 2, 3]).
%% true ;
%% false.
%% ?- sufixo([2, 3], [1, 2, 2]).
%% false.
%% ?- sufixo(5, 5).
%% true ;
%% false.
%% ?- sufixo(X, [1, 2, 3]).
%% X = [1, 2, 3] ;
%% X = [2, 3] ;
%% X = [3] ;
%% X = [] ;
%% false.
%% ?- sufixo([2, 3], Y).
%% Y = [2, 3] ;
%% Y = [_G247, 2, 3] ;
%% Y = [_G247, _G250, 2, 3] ;
%% Y = [_G247, _G250, _G253, 2, 3] ;
%% Y = [_G247, _G250, _G253, _G256, 2, 3] ;
%% Y = [_G247, _G250, _G253, _G256, _G259, 2, 3]
%% ?- sufixo(X, Y).
%% X = Y ;
%% Y = [_G250|X] ;
```

```
%% Y = [_G250, _G253|X] ;
%% Y = [_G250, _G253, _G256|X] ;
%% Y = [_G250, _G253, _G256, _G259|X] ;
%% Y = [_G250, _G253, _G256, _G259, _G262|X]


/*
  sublista(Sub, Lista) :- tem o valor verdadeiro se Sub for uma
  sublista da lista Lista.
*/
%%% definicao recursiva
sublista(Xs, Ys) :- prefixo(Xs, Ys).
sublista(Xs, [_|Ys]) :- sublista(Xs, Ys).
                        %aqui nao podia ser prefixo, se fosse so considerava
                        %sublistas a comecar na primeira e segunda posicoes

%%% sufixo de um prefixo
sublista2(Xs, Ys) :- prefixo(Ps, Ys), sufixo(Xs, Ps).

%%% prefixo de um sufixo
sublista3(Xs, Ys) :- prefixo(Xs, Ss), sufixo(Ss, Ys).

%% ?- sublista([2, 3], [1, 2, 3, 4]).
%% true ;
%% false.
%% ?- sublista([2, 3], [1, 2, 2, 4]).
%% false.
%% ?- sublista(X, [1, 2, 3, 4]).
%% X = [] ;
%% X = [1] ;
%% X = [1, 2] ;
%% X = [1, 2, 3] ;
%% X = [1, 2, 3, 4] ;
%% X = [] ;
%% X = [2] ;
%% X = [2, 3] ;
%% X = [2, 3, 4] ;
%% X = [] ;
%% X = [3] ;
%% X = [3, 4] ;
%% X = [] ;
%% X = [4] ;
%% X = [] ;
%% false.
%% ?- sublista([2, 3], Y).
%% Y = [2, 3|_G251] ;
%% Y = [_G247, 2, 3|_G254] ;
%% Y = [_G247, _G250, 2, 3|_G257] ;
%% Y = [_G247, _G250, _G253, 2, 3|_G260] ;
%% Y = [_G247, _G250, _G253, _G256, 2, 3|_G263]
%% ?- sublista(X, Y).
%% X = [] ;
%% X = [_G661],
%% Y = [_G661|_G665] ;
%% X = [_G661, _G667],
%% Y = [_G661, _G667|_G671] ;
```

```
%% X = [_G661, _G667, _G673],
%% Y = [_G661, _G667, _G673|_G677] ;
%% X = [_G661, _G667, _G673, _G679],
%% Y = [_G661, _G667, _G673, _G679|_G683] ;
%% X = [_G661, _G667, _G673, _G679, _G685],
%% Y = [_G661, _G667, _G673, _G679, _G685|_G689]

/*
  junta(Xs, Ys, Zs) :- a lista Zs e o resultado de concatenar as listas Xs e Ys.
*/
junta([], Ys, Ys).
junta([X|Xs], Ys, [X|Zs]) :- junta(Xs, Ys, Zs).

%% ?- junta([1, 2], [3, 4], [1, 2, 3, 4]).
%% true.
%% ?- junta([1, 2], [3, 3], [1, 2, 3, 4]).
%% false.
%% ?- junta([1, 2], [3, 4], Z).
%% Z = [1, 2, 3, 4].
%% ?- junta([1, 2], Y, [1, 2, 3, 4]).
%% Y = [3, 4].
%% ?- junta(X, [3, 4], [1, 2, 3, 4]).
%% X = [1, 2] ;
%% false.
%% ?- junta(X, Y, [1, 2, 3, 4]).
%% X = [],
%% Y = [1, 2, 3, 4] ;
%% X = [1],
%% Y = [2, 3, 4] ;
%% X = [1, 2],
%% Y = [3, 4] ;
%% X = [1, 2, 3],
%% Y = [4] ;
%% X = [1, 2, 3, 4],
%% Y = [] ;
%% false.
%% ?- junta(X, [3, 4], Z).
%% X = [],
%% Z = [3, 4] ;
%% X = [_G267],
%% Z = [_G267, 3, 4] ;
%% X = [_G267, _G273],
%% Z = [_G267, _G273, 3, 4] ;
%% X = [_G267, _G273, _G279],
%% Z = [_G267, _G273, _G279, 3, 4] ;
%% X = [_G267, _G273, _G279, _G285],
%% Z = [_G267, _G273, _G279, _G285, 3, 4]
%% ?- junta(X, Y, Z).
%% X = [],
%% Y = Z ;
%% X = [_G270],
%% Z = [_G270|Y] ;
%% X = [_G270, _G276],
%% Z = [_G270, _G276|Y] ;
%% X = [_G270, _G276, _G282],
%% Z = [_G270, _G276, _G282|Y] ;
```

```
%% X = [_G270, _G276, _G282, _G288],
%% Z = [_G270, _G276, _G282, _G288|Y]


/*
  Redefinir os predicados membro, prefixo, sufixo e sublista
  em termos do predicado junta.
*/
membro2(X, Ys) :- junta(_, [X|_], Ys).

prefixo2(Xs, Ys) :- junta(Xs, _, Ys).

sufixo2(Xs, Ys) :- junta(_, Xs, Ys).

%%% prefixo de um sufixo
sublista4(Xs, AsXsBs) :- junta(_, XsBs, AsXsBs), junta(Xs, _, XsBs).

%%% sufixo de um prefixo
sublista5(Xs, AsXsBs) :- junta(AsXs, _, AsXsBs), junta(_, Xs, AsXs).


/*
  seguidos(X, Y, Zs) :- tem o valor verdadeiro se X e Y aparecerem
  seguidos na lista Zs.
*/
seguidos(X, Y, Zs) :- junta(_, [X, Y|_], Zs).

%% ?- seguidos(1, 2, [1, 2, 3]).
%% true ;
%% false.
%% ?- seguidos(1, 2, [2, 2, 3]).
%% false.
%% ?- seguidos(1, Y, [1, 2, 3]).
%% Y = 2 ;
%% false.
%% ?- seguidos(X, Y, [1, 2, 3]).
%% X = 1,
%% Y = 2 ;
%% X = 2,
%% Y = 3 ;
%% false.
%% ?- seguidos(X, Y, Z).
%% Z = [X, Y|_G271] ;
%% Z = [_G273, X, Y|_G271] ;
%% Z = [_G273, _G279, X, Y|_G271] ;
%% Z = [_G273, _G279, _G285, X, Y|_G271] ;
%% Z = [_G273, _G279, _G285, _G291, X, Y|_G271]


/*
  ultimo(X, Xs) :- tem o valor verdadeiro se X for o ultimo
  elemento da lista Xs.
*/
ultimo(X, Xs) :- junta(_, [X], Xs).

%% ?- ultimo(3, [1, 2, 3]).
```

```
%% true ;
%% false.
%% ?- ultimo(2, [1, 2, 3]).
%% false.
%% ?- ultimo(X, [1, 2, 3]).
%% X = 3 ;
%% false.
%% ?- ultimo(3, Y).
%% Y = [3] ;
%% Y = [_G238, 3] ;
%% Y = [_G238, _G244, 3] ;
%% Y = [_G238, _G244, _G250, 3] ;
%% Y = [_G238, _G244, _G250, _G256, 3]
%% true
%% ?- ultimo(X, Y).
%% Y = [X] ;
%% Y = [_G253, X] ;
%% Y = [_G253, _G259, X] ;
%% Y = [_G253, _G259, _G265, X] ;
%% Y = [_G253, _G259, _G265, _G271, X]


/*
  inverte(Xs, Ys) :- tem o valor verdadeiro se Ys for uma lista que
  contem os elementos de Xs pela ordem inversa da qual eles aparecem na
  lista original.
*/

%%% versao inicial
inverteInicial([], []).
inverteInicial([X|Xs], Zs) :- inverteInicial(Xs, Ys), junta(Ys, [X], Zs).

%%% inverte com is --- nao funciona, pois o is deve ser usado com numeros
inverteIs([], []).
inverteIs([X|Xs], Zs) :- inverteIs(Xs, Ys), Zs is [X|Ys].

%%% versao com acumulador
inverte(Xs, Ys) :- inverte(Xs, [], Ys).

inverte([X|Xs], Acc, Ys) :- inverte(Xs, [X|Acc], Ys).
inverte([], Ys, Ys).

%% ?- inverteInicial([1, 2, 3], [3, 2, 1]).
%% true.
%% ?- inverteInicial([1, 1, 1], [3, 2, 1]).
%% false.
%% ?- inverteInicial(X, [3, 2, 1]).
%% X = [1, 2, 3] ;
%%   C-c C-c
%% Action (h for help) ? h
%% Options:
%% a:                abort      b:                break
%% c:                continue   e:                exit
%% g:                goals      t:                trace
%% h (?):            help
%% Action (h for help) ? a:
```

```
%% abort
%% % Execution Aborted
%% ?- inverteInicial(X, [3, 2, 1]).
%% X = [1, 2, 3]
%% ?- inverteInicial([1, 2, 3], Y).
%% Y = [3, 2, 1].
%% ?- inverteInicial(X, Y).
%% X = Y, Y = [] ;
%% X = Y, Y = [_G295] ;
%% X = [_G253, _G256],
%% Y = [_G256, _G253] ;
%% X = [_G253, _G256, _G259],
%% Y = [_G259, _G256, _G253] ;
%% X = [_G253, _G256, _G259, _G262],
%% Y = [_G262, _G259, _G256, _G253]
%% ?- inverte([1, 2, 3], [3, 2, 1]).
%% true.
%% ?- inverte([1, 1, 1], [3, 2, 1]).
%% false.
%% ?- inverte(X, [3, 2, 1]).
%% ERROR: Out of local stack
%% ?- inverte([1, 2, 3], Y).
%% Y = [3, 2, 1].
%% ?- inverte(X, Y).
%% ERROR: Out of local stack


/*
  comprimento(Xs, N) :- tem o valor verdadeiro se N for o comprimento da lista Xs.
*/
comprimento([], 0).
comprimento([_|Xs], s(N)) :- comprimento(Xs, N).

%% ?- comprimento([], X).
%% X = 0.
%% ?- comprimento([1, 2, 3], s(s(s(0)))).
%% true.
%% ?- comprimento([1, 2, 3], s(s(0))).
%% false.
%% ?- comprimento([1, 2, 3], X).
%% X = s(s(s(0))).
%% ?- comprimento(X, s(s(s(0)))).
%% X = [_G250, _G253, _G256].


/*
  repete(Xs, XsXs) :- que tem o valor verdadeiro se cada elemento de
  Xs aparece repetido em XsXs.
*/
repete([], []).
repete([X|Xs], [X, X|Ys]) :- repete(Xs, Ys).

%% ?- repete([1, 2], [1, 1, 2, 2]).
%% true.
%% ?- repete([1, 2], [1, 2]).
%% false.
```

```
%% ?- repete([1, 2], Y).
%% Y = [1, 1, 2, 2].
%% ?- repete(X, [1, 1, 2, 2]).
%% X = [1, 2].
%% ?- repete(X, [1, 1, 2]).
%% false.
%% ?- repete(X, Y).
%% X = [], Y = [] ;
%% X = [_G256],
%% Y = [_G256, _G256] ;
%% X = [_G256, _G265],
%% Y = [_G256, _G256, _G265, _G265] ;
%% X = [_G256, _G265, _G274],
%% Y = [_G256, _G256, _G265, _G265, _G274, _G274] ;
%% X = [_G256, _G265, _G274, _G283],
%% Y = [_G256, _G256, _G265, _G265, _G274, _G274, _G283, _G283]
```

14 Prolog — Operadores Pré-definidos

Sumário:

- Operadores pré-definidos
- Acrescentar/remover cláusulas
- Inspecção da estrutura dos termos
- Predicados *soluções-todas*

Resumo:

Operadores pré-definidos

Operadores relacionais

- `=` — Predicado de unificação: `<t1> = <t2>` tem sucesso se `t1` e `t2` podem ser unificados. A unificação de duas variáveis não instanciadas faz com que elas partilhem a mesma representação interna.
- `\=` — Negação do predicado de unificação.
- `==` — Predicado de identidade: `<t1> == <t2>` tem sucesso se `t1` e `t2` são idênticos (sem instanciar variáveis).
- `\==` — Negação do predicado de identidade, indica que dois termos têm que ser diferentes.

Operações numéricas

- `<t1> + <t2>` — soma.
- `<t1> - <t2>` — subtracção.
- `-<t1>` — simétrico.
- `<t1> * <t2>` — multiplicação.
- `<t1> / <t2>` — divisão.
- `(/ <t1>)` — inverso.
- `<t1> ** <t2>` — potência.
- `<t1> // <t2>` — divisão inteira.
- `<t1> mod <t2>` — resto da divisão inteira.
- `(round <t1>)` — arredondamento para o inteiro mais próximo.
- `(sqrt <t1>)` — raiz quadrada.
- `(abs <t1>)` — valor absoluto.

Operadores relacionais numéricos

- =:= — Igualdade aritmética.
- =\= — Negação da igualdade aritmética.
- < — Menor.
- > — Maior.
- =< — Menor ou igual.
- >= — Maior ou igual.

O operador de atribuição `<variável> is <termo>` avalia o termo e liga o seu valor à variável. O resultado da sua avaliação é verdadeiro. Com a atribuição, já não é possível usar qualquer argumento dos predicados como variável, apenas aqueles que estavam previstos.

Instruções de escrita `write(<t1>)`, `nl`, entre outras.

A representação `predicado(+A, +B, C, D)` significa que o `predicado` deve ser usado com os dois argumentos `A` e `B` instanciados (argumentos de entrada) e dois argumentos `C` e `D` como variáveis (argumentos de saída).

Acrescentar/remover cláusulas

`dynamic[predicado]` — um predicado dinâmico é um predicado ao qual podem ser acrescentadas ou removidas cláusulas ao longo da execução do programa.

`assert` — adiciona uma cláusula a um predicado na base de dados. As cláusulas podem ser adicionadas no início ou no fim da lista de cláusulas do predicado.
`assert(+Termo)` — adiciona a cláusula como a última do predicado.
`asserta(+Termo)` — adiciona a cláusula como a primeira do predicado.

`retract(+Termo)` — quando o `Termo` é um átomo ou um termo, unifica com o primeiro facto ou cláusula unificável da base de dados, que é removido.
`retractall(+Cabeca)` — todas as cláusulas na base de dados cuja cabeça unifica com `Cabeca` são removidas.

```
?- assert(maisAlto(nuno, maria)).
true.
?- assert(maisAlto(nuno, ana)).
true.
?- assert((maisBaixo(X, Y) :- maisAlto(Y, X))).
true.
?- maisBaixo(X, nuno).
X = maria ;
X = ana.
?- retractall(maisBaixo(X, Y)).
true.
?- maisBaixo(maria, nuno).
```

```
false.
?- retract(maisAlto(nuno, ana)).
true.
?-  maisAlto(nuno, X).
X = maria.
?-
```

Inspecção da estrutura dos termos

`integer(X),real(X),atom(X),compound(X)`

`number(X)` é o mesmo que `integer(X)` ou `real(X)`.

`atomic(X)` é o mesmo que `atom(X)` ou `number(X)`.

`var(Term),nonvar(Term)`

Predicados *soluções-todas*

Predicados *soluções-todas* são predicados que retornam todas as instâncias que são resposta de um objectivo.

`findall(+Template, +Objectivo, -Saco)` — cria uma lista com as instanciações obtidas para `Template` fazendo retrocessos sucessivos sobre o `Objectivo` e unifica o resultado com `Saco`. Tem sucesso com uma lista vazia se o `Objectivo` não tiver soluções. É equivalente ao `bagog` com todas as variáveis usando o operador existencial (`^`), excepto no facto que o `bagof` falha quando não existem soluções.

`bagof(+Template, +Objectivo, -Saco)` — unifica `Saco` com as alternativas de `Template`, se o `Objectivo` tiver variáveis livres para além das partilhadas com o `Template`. A notação `Variavel^Objectivo` diz ao `bagof` para não ligar a `Variável` no `Objectivo`. Falha se o `Objectivo` não tiver soluções.

`setof(+Template, +Objectivo, -Conjunto)` — equivalente ao `bagof`, mas ordena o resultado usando `sort/2` para obter uma lista ordenada sem duplicados (um conjunto ordenado).

Exercício 14.1
Considere o seguinte programa para calcular o factorial de um número.

```
/*
  factorial1(N, F) :- F e o factorial de N.
*/
factorial1(0, 1).
factorial1(N, F) :- N1 is N-1, factorial1(N1, F1), F is N*F1.
```

1. Qual a sua resposta ao objectivo `factorial1(3, X)`?
2. O que acontece se pedirmos uma segunda solução? Escreva um programa que resolva o problema.
3. Qual a sua resposta ao objectivo `factorial1(X, 6)`? Como resolver o problema?

Resposta:

1. ```
   ?- factorial1(3, X).
   X = 6
   ```
2. ```
   ?- factorial1(3, X).
   X = 6 ;
   ERROR: Out of local stack
   ```
   Neste caso, o programa entra em ciclo infinito. Este problema pode ser resolvido usando uma outra definição para o factorial, por exemplo:

   ```
   /*
     factorial(N, F) :- F e o factorial de N.
   */
   factorial(N, F) :- factorial(N, 1, F).
   factorial(N, T, F) :- N > 0, T1 is T*N, N1 is N-1, factorial(N1, T1, F).
   factorial(0, F, F).
   ```

   Com esta definição, o Prolog nem sequer permite que seja pedida outra solução:

   ```
   ?- factorial(3, X).

   X = 6.
   ?-
   ```
3. A resposta de ambos os programas a este objectivo é um erro:

   ```
   ?- factorial1(X, 6).
   ERROR: is/2: Arguments are not sufficiently instantiated
   ^  Exception: (8) _L136 is _G170-1 ? a
   % Execution Aborted
   ?- factorial(X, 6).
   ERROR: >/2: Arguments are not sufficiently instantiated
   ^  Exception: (10) _G173>0 ? a
   % Execution Aborted
   ?-
   ```

   Neste caso, se quisermos ter um programa que determina de que número é que um valor é o factorial, temos que escrever outro programa específico para o efeito, pois a utilização do `is` e dos operadores matemáticos obriga a que a computação seja feita apenas num sentido.

Exercício 14.2
Escreva o predicado `comp(L, C)`, que tem o valor verdadeiro se `C` é o comprimento da lista `L`.

1. Gerando um processo recursivo.
2. Gerando um processo iterativo.

Resposta:

1.
```
/*
  comp(L, C) :- C e o comprimento da lista L.
*/
comp([], 0).
comp([_|Xs], C) :- comp(Xs, Cx), C is Cx+1.

%% O caso base aparece antes para dar uma resposta a comp(X, 2).
%% ?- comp([], X).
%% X = 0.
%% ?- comp([1, 2, 3], X).
%% X = 3.
%% ?- comp([1, 2, 3], 5).
%% false.
%% ?- comp(X, 2).
%% X = [_G241, _G244] ;
%%   C-c C-c
%% Action (h for help) ? a
%% abort
%% % Execution Aborted
%% ?-
```
2.
```
/*
  comp1(L, C) :- C e o comprimento da lista L.
  Usando um acumulador.
*/
comp1(L, C) :- comp1(L, 0, C).

comp1([], Acc, Acc).
comp1([_|Xs], Acc, C) :- Acc1 is Acc+1, comp1(Xs, Acc1, C).

%% O caso base aparece antes para dar uma resposta a comp1(X, 2).
%% ?- comp1([], X).
%% X = 0.
%% ?- comp1([1, 2, 3], X).
%% X = 3.
%% ?- comp1([1, 2, 3], 5).
%% false.
%% ?- comp1(X, 2).
%% X = [_G241, _G247] ;
%% ERROR: Out of global stack
%% ?-
```

Exercício 14.3
Escreva o predicado `somalista(Xs, S)`, que tem o valor verdadeiro se `S` corresponde à soma de todos os elementos da lista de inteiros `Xs`.

1. Gerando um processo recursivo.
2. Gerando um processo iterativo.

Resposta:

1.
```
/*
  somalista(Xs, S) :- tem o valor verdadeiro se S corresponde
  a soma de todos os elementos da lista de inteiros Xs.
*/
somalista([X|Xs], S) :- somalista(Xs, S1), S is S1+X.
somalista([], 0).

%% ?- somalista([1, 2, 3, 4], X).
%% X = 10.
%% ?- somalista([1, -22, 3*4, 4], X).
%% X = -5.
%% ?- somalista([1, -22, 3*4, 4], 4).
%% false.
%% ?-
```

2.
```
/*
  somalista(Xs, S) :- tem o valor verdadeiro se S corresponde
  a soma de todos os elementos da lista de inteiros Xs.
  Usando um acumulador.
*/
somalista1(Xs, S) :- somalista1(Xs, 0, S).

somalista1([X|Xs], Acc, S) :- Acc1 is Acc + X, somalista1(Xs, Acc1, S).
somalista1([], S, S).

%% ?- somalista1([1, 2, 3, 4], X).
%% X = 10.
%% ?- somalista1([1, -22, 3*4, 4], X).
%% X = -5.
%% ?- somalista1([1, -22, 3*4, 4], 4).
%% false.
%% ?-
```

Exercício 14.4
Escreva o predicado `remove(Xs, X, Ys)`, que tem o valor verdadeiro se `Ys` resulta de remover todas as ocorrências de `X` da lista `Xs`.

Resposta:

```
/*
  remove(Xs, X, Ys) :- tem o valor verdadeiro se Ys resulta de
  remover todas as ocorrencias de X da lista Xs.
*/
remove([X|Xs], X, Ys) :- remove(Xs, X, Ys).
remove([X|Xs], Z, [X|Ys]) :- X \= Z, remove(Xs, Z, Ys).
remove([], _, []).

%% ?- remove([], 3, X).
%% X = [].
```

```
%% ?- remove([1, 2, 3, 2], 2, X).
%% X = [1, 3] ;
%% false.
%% ?- remove([1, 2, 3, 2], 4, X).
%% X = [1, 2, 3, 2].
%% ?- remove([1, 2, 3, 2], 2, [1]).
%% false.
%% ?-

%% A condicao X \= Z e necessaria senao dava varias solucoes como
%% resposta, em que removia todas, todas menos uma, todas menos duas,
%% ..., ate nao remover nenhuma das ocorrencias de X.  Por exemplo,

removeErrado([X|Xs], X, Ys) :- removeErrado(Xs, X, Ys).
removeErrado([X|Xs], Z, [X|Ys]) :- removeErrado(Xs, Z, Ys).
removeErrado([], _, []).

%% ?- removeErrado([1, 2, 3, 2], 2, X).
%% X = [1, 3] ;
%% X = [1, 3, 2] ;
%% X = [1, 2, 3] ;
%% X = [1, 2, 3, 2].
%% ?-


%% Na segunda clausula e necessario adicionar X ao resultado na cabeca
%% e nao no corpo da clausula porque se o fizesse no corpo nunca iria
%% emparelhar com o caso base da recursao.  Ou seja, a lista do
%% resultado nao seria vazia quando a lista inicial o fosse (so dava o
%% resultado certo nos casos em que este correspondesse a lista vazia).

removeErrado2([X|Xs], X, Ys) :- removeErrado2(Xs, X, Ys).
removeErrado2([X|Xs], Z, Ys) :- X \= Z, removeErrado2(Xs, Z, [X|Ys]).
removeErrado2([], _, []).

%% ?- removeErrado2([], 3, X).
%% X = [].
%% ?- removeErrado2([2, 2, 2, 2], 2, X).
%% X = [] ;
%% false.
%% ?- removeErrado2([1, 2, 3, 2], 2, X).
%% false.
%% ?- removeErrado2([1, 2, 3, 2], 4, X).
%% false.
%% ?- removeErrado2([1, 2, 3, 2], 2, [1, 3]).
%% false.
%% ?-
```

Exercício 14.5
Escreva o predicado `escreveLista(Xs)`, que escreve todos os elementos da lista `Xs`, um por linha, e a mensagem `Fim da lista.` no fim da lista.

Resposta:

```
/*
```

```
  escreveLista(Xs) :- escreve todos os elementos da lista Xs,
  um por linha.
*/
escreveLista([X|Xs]) :- write(X), nl, escreveLista(Xs).
escreveLista([]) :- write('Fim da lista.'), nl.

%% ?- escreveLista([1, 2, 3]).
%% 1
%% 2
%% 3
%% Fim da lista.
%% true.
%% ?- escreveLista([ola, 'ola bom dia', "ola"]).
%% ola
%% ola bom dia
%% [111, 108, 97]
%% Fim da lista.
%% true.
%% ?-
```

Exercício 14.6
Suponha que queremos definir um predicado que permita indicar que uma pessoa é um avô português de alguém. Supondo que temos uma base de dados em Prolog com os predicados `portugues(X)` e `avo(X,Y)`, podemos definir um predicado `avoPortugues` de uma das seguintes maneiras:

- Definição 1: `avoPortugues1(X,Y) :- portugues(X), avo(X,Y).`
- Definição 2: `avoPortugues2(X,Y) :- avo(X,Y), portugues(X).`

Diga se alguma das definições é mais eficiente do que a outra, em que condições, e explique porquê.

Resposta:

Em primeiro lugar convém notar que em termos lógicos as duas definições são equivalentes e devem dar os mesmos resultados.

Se tanto `X` como `Y` estiverem instanciados, por exemplo se estivermos interessados em saber se o Nuno é um avô português do Rui (`avoPortugues(Nuno, Rui)`), não há grandes diferenças em termos de eficiência, pois são feitos os mesmos pedidos à base de dados do Prolog.

Se `X` tiver um valor mas `Y` for variável, por exemplo se estivermos interessados em saber de quem é que o Nuno é um avô português (`avoPortugues(Nuno, Y)`), a primeira será mais eficiente se o `Nuno` não for português porque não precisa de procurar os netos, e ambas são igualmente eficientes se o `Nuno` for português.

Se `X` for variável mas `Y` tiver um valor, por exemplo se estivermos interessados em saber quem são os avôs portugueses do Rui (`avoPortugues(X, Rui)`, a segunda definição é mais eficiente do que a primeira. Na primeira definição, o Prolog vai emparelhar `X` com todos os portugueses e só depois é que vai verificar se cada um deles é ou não avô do `Rui`. Na segunda definição, primeiro o Prolog determina quais são os avôs do `Rui` e depois vai verificar se são portugueses. Como o número de avós de alguém é muito menor do que o número de portugueses, a segunda alternativa é muito mais eficiente do que a primeira.

Se tanto `X` como `Y` forem variáveis, convinha saber se há mais portugueses ou avôs na base de dados, e testar primeiro os que estiverem em menor número.

Se soubermos em que condições é que o predicado vai ser chamado com mais frequência, podemos escolher a implementação que melhor se adequa à nossa aplicação.

Exercício 14.7
Suponha que tem uma base de dados que indica as notas que os alunos tiveram nas várias disciplinas (`nota(Nome, Disciplina, Nota)`) e quais os alunos inscritos nas várias disciplinas (`inscrito(Nome, Disciplina)`).

Escreva um programa que permite lançar notas de alunos às disciplinas a que eles estão inscritos e que determina a seguinte informação:

- Lista dos alunos com pelo menos uma nota superior ou igual a um dado valor.
- Média das notas de uma disciplina.
- Média das notas de um aluno.

Resposta:

Ficheiro `bdAlunos.pl`

```
% Para poder usar assert, retract, um predicado tem que ser dynamic
:- dynamic nota/3.
:- dynamic inscrito/2.
/*
  nota(Nome, Disciplina, Nota) :- o aluno chamado Nome fez a Disciplina
  com a Nota indicada.
*/

nota(maria, lp, 15).
nota(maria, fp, 16).
nota(maria, iaed, 13).
nota(maria, bd, 13).
nota(maria, comp, 17).

nota(pedro, lp, 16).
nota(pedro, fp, 15).
nota(pedro, iaed, 14).
nota(pedro, bd, 14).
nota(pedro, comp, 16).

nota(nuno, lp, 12).
nota(nuno, fp, 13).
nota(nuno, iaed, 15).
nota(nuno, bd, 13).
nota(nuno, comp, 15).

/*
  inscrito(Nome, Disciplina) :- o aluno chamado Nome esta inscrito na
  Disciplina.
*/
```

```
inscrito(maria, as).
inscrito(maria, es).

inscrito(pedro, as).
inscrito(pedro, es).
inscrito(pedro, qs).
```

Ficheiro `alunos.pl`

```
:- consult(bdAlunos).

/*
  lancaNota(Aluno, Disciplina, Nota) :- lanca a Nota de um Aluno a uma Disciplina
  em que ele esta inscrito.
*/
lancaNota(Aluno, Disciplina, Nota) :-
        inscrito(Aluno, Disciplina),
        retract(inscrito(Aluno, Disciplina)),
        actualizaNota(Aluno, Disciplina, Nota);
        write('Impossivel lancar nota porque '),
        write(Aluno),
        write(' nao esta inscrito(a) a '),
        write(Disciplina),
        write('.').


/*
  actualizaNota(Aluno, Disciplina, Nota) :- actualiza a Nota de um Aluno a uma
  Disciplina.
*/
actualizaNota(Aluno, Disciplina, Nota) :-
        nota(Aluno, Disciplina, NotaAnterior),
        NotaAnterior >= Nota,
        write('O aluno ja tinha uma nota melhor.');
        retractall(nota(Aluno, Disciplina, _)),
        %% o retractall nao falha, mesmo que nao exista uma nota anterior
        assert(nota(Aluno, Disciplina, Nota)).


/*
  listaAlunosNotaSuperiorA(Nota, ListaAlunos) :- determina a lista
  dos alunos com pelo menos uma nota superior a um dado valor.
*/
listaAlunosNotaSuperiorA(Nota, ListaAlunos) :-
        setof(Aluno, alunoComNota(Aluno, Nota), ListaAlunos).
        %% usa-se setof porque nao se quer que os alunos aparecam repetidos


/*
  alunoComNota(Aluno, Nota) :- o Aluno tem pelo menos uma nota
  superior a Nota.
*/
alunoComNota(Aluno, Nota) :-
        nota(Aluno, _, Nota2),
```

```
        Nota2 >= Nota.


/*
  escreveLista(Lista) :- escreve os elementos da Lista, um por linha.
*/
escreveLista([X|Xs]) :- write(X), nl, escreveLista(Xs).
escreveLista([]) :- write('Fim da lista').


/*
  mediaAluno(Aluno) :- determina a media das notas do Aluno.
*/
mediaAluno(Aluno) :-
        bagof(Nota, Disciplina^nota(Aluno, Disciplina, Nota), ListaNotas),
        %% usa-se bagof porque todas as notas contam, mesmo que sejam repetidas
        escreveMedia(ListaNotas, 'A media das notas do(a) ', Aluno).


/*
  mediaDisciplina(Disciplina) :- determina a media das notas dos alunos
  que ja fizeram a disciplina.
*/
mediaDisciplina(Disciplina) :-
        bagof(Nota, Nome^nota(Nome, Disciplina, Nota), ListaNotas),
        escreveMedia(ListaNotas, 'A media das notas de ', Disciplina).


/*
  escreveMedia(Lista, Mensagem, Objecto) :- determina a media dos numeros
  da lista e escreve-a.
*/
escreveMedia(Lista, Mensagem, Objecto) :-
        calculaMedia(Lista, Media),
        write(Mensagem),
        write(Objecto),
        write(' e '),
        write(Media),
        write('.'),
        nl.


/*
  calculaMedia(Lista, Media) :- determina a media dos numeros da lista.
*/
calculaMedia(Lista, Media) :-
        contaESoma(Lista, 0, 0, Elems, Soma),
        Media is Soma/Elems.


/*
  contaESoma(Lista, AccElems, AccSoma, Elems, Soma) :-
  conta e soma os numeros da lista.
*/
contaESoma([X|Xs], AccElems, AccSoma, Elems, Soma) :-
        AccElems1 is AccElems + 1,
```

```
        AccSoma1 is AccSoma + X,
        contaESoma(Xs, AccElems1, AccSoma1, Elems, Soma).
contaESoma([], AccElems, AccSoma, AccElems, AccSoma).
```

Exemplo de utilização

```
% library(swi_hooks) compiled into pce_swi_hooks 0.00 sec, 3,928 bytes
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 5.10.4)
Copyright (c) 1990-2011 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?- ['alunos.pl'].
%  bdAlunos compiled 0.00 sec, 4,320 bytes
% alunos.pl compiled 0.00 sec, 11,768 bytes
true.

?-  bagof(Nota, nota(maria, Disciplina, Nota), ListaNotas).
Disciplina = bd,
ListaNotas = [13] ;
Disciplina = comp,
ListaNotas = [17] ;
Disciplina = fp,
ListaNotas = [16] ;
Disciplina = iaed,
ListaNotas = [13] ;
Disciplina = lp,
ListaNotas = [15].

?- setof(Nota, nota(maria, Disciplina, Nota), ListaNotas).
Disciplina = bd,
ListaNotas = [13] ;
Disciplina = comp,
ListaNotas = [17] ;
Disciplina = fp,
ListaNotas = [16] ;
Disciplina = iaed,
ListaNotas = [13] ;
Disciplina = lp,
ListaNotas = [15].

?- findall(Nota, nota(maria, Disciplina, Nota), ListaNotas).
ListaNotas = [15, 16, 13, 13, 17].

?- bagof(Nota, Disciplina^nota(maria, Disciplina, Nota), ListaNotas).
ListaNotas = [15, 16, 13, 13, 17].

?- setof(Nota, Disciplina^nota(maria, Disciplina, Nota), ListaNotas).
ListaNotas = [13, 15, 16, 17].

?- findall(Nota, Disciplina^nota(maria, Disciplina, Nota), ListaNotas).
```

```
ERROR: fa_loop/5: Undefined procedure: (^)/2
ERROR:   ^/2 can only appear as the 2nd argument of setof/3 and bagof/3
?- bagof(Nota, nota(maria, _, Nota), ListaNotas).
ListaNotas = [13] ;
ListaNotas = [17] ;
ListaNotas = [16] ;
ListaNotas = [13] ;
ListaNotas = [15].

?- setof(Nota, nota(maria, _, Nota), ListaNotas).
ListaNotas = [13] ;
ListaNotas = [17] ;
ListaNotas = [16] ;
ListaNotas = [13] ;
ListaNotas = [15].

?- findall(Nota, nota(maria, _, Nota), ListaNotas).
ListaNotas = [15, 16, 13, 13, 17].

?- bagof(Nota, nota(ana, _, Nota), ListaNotas).
false.

?- setof(Nota, nota(ana, _, Nota), ListaNotas).
false.

?- findall(Nota, nota(ana, _, Nota), ListaNotas).
ListaNotas = [].

?- listaAlunosNotaSuperiorA(16, ListaAlunos).
ListaAlunos = [maria, pedro].

?- lancaNota(pedro, xpto, 15).
Impossivel lancar nota porque pedro nao esta inscrito(a) a xpto.
true.

?- lancaNota(pedro, as, 15).
true

?- nota(pedro, as, X).
X = 15.

?- assert(inscrito(pedro, as)).
true.

?- lancaNota(pedro, as, 13).
O aluno ja tinha uma nota melhor.
true

?- nota(pedro, as, X).
X = 15.

?- setof((Nota, Disciplina), nota(nuno, Disciplina, Nota), ListaNotas).
ListaNotas = [ (12, lp), (13, bd), (13, fp), (15, comp), (15, iaed)].

?- bagof((Nota, Disciplina), nota(nuno, Disciplina, Nota), ListaNotas).
ListaNotas = [ (12, lp), (13, fp), (15, iaed), (13, bd), (15, comp)].
```

```
?- setof(Nota, Disciplina^nota(nuno, Disciplina, Nota), ListaNotas).
ListaNotas = [12, 13, 15].

?- bagof(Nota, Disciplina^nota(nuno, Disciplina, Nota), ListaNotas).
ListaNotas = [12, 13, 15, 13, 15].

?-  mediaAluno(nuno).
A media das notas do(a) nuno e 13.6.
true.

?- bagof(Nota, Aluno^nota(Aluno, iaed, Nota), ListaNotas).
ListaNotas = [13, 14, 15].

?- mediaDisciplina(iaed).
A media das notas de iaed e 14.
true.

?- halt.

Process prolog finished
```

15 Prolog — Corte; Negação

Sumário:

- Corte
- Negação

Resumo:

Corte

O operador de corte (`!`) tem sempre sucesso e impede o retrocesso na árvore de refutação para além do ponto onde é resolvido, comprometendo-se com todas as escolhas feitas desde que o objectivo inicial unificou com a cabeça da cláusula onde o corte ocorre. Serve para impedir que o Prolog explore ramos das árvores de refutação que se sabe que não devem ser explorados.

- Um corte elimina todas as cláusulas abaixo dele. Um objectivo unificado com uma cláusula contendo um corte que teve sucesso não consegue produzir soluções usando cláusulas que ocorram abaixo dessa cláusula.
- O corte elimina todas as soluções alternativas para a conjunção de objectivos à sua esquerda. Por exemplo, uma conjunção seguida de um corte produzirá apenas uma solução.
- O corte não afecta os objectivos à sua direita na cláusula. Estes podem produzir mais do que uma solução em caso de retrocesso. No entanto, quando esta conjunção falha, a procura continua a partir da última alternativa anterior à escolha da cláusula contendo o corte.

Os *cortes verdes* servem para expressar determinismo, por exemplo quando um programa tem várias cláusulas que se sabe que são mutuamente exclusivas, isto é, que quando o fluxo de execução passa um determinado teste não vai passar os testes das outras cláusulas.

Os *cortes vermelhos* são usados para aumentar a eficiência dos programas, eliminando soluções alternativas que à partida não se quer que sejam geradas. Mas alteram os resultados obtidos, e por isso devem ser usados ainda com mais cuidado.

Exemplos de corte

Se o Prolog encontra um corte numa regra, ele não vai retroceder nas escolhas que já fez para o objectivo que está a provar. Isto é ilustrado pelo programa seguinte:

```
a1(X, Y) :- b1(X), !, c1(Y).
b1(1).
b1(2).
b1(3).
c1(1).
c1(2).
c1(3).
```

Se perguntarmos `a1(Q, R)`, o Prolog pode dar várias respostas:

```
?- a1(Q, R).
Q = R, R = 1 ;
Q = 1,
R = 2 ;
Q = 1,
R = 3.

?-
```

A única opção considerada para `Q` é `1`, mas são consideradas todas as alternativas para `R`. Quando o Prolog começa a provar o objectivo, tenta provar `a1(Q, R)` usando a primeira cláusula do programa. Para isso, precisa de provar `b1(Q)`, que tem sucesso com `Q = 1`. Depois encontra o corte e `Q = 1` passa a ser a única opção possível para `Q`. Depois continua com `c1(R)`. Primeiro, `R = 1` e o objectivo é concluído. Quando o utilizador tecla `;`, o Prolog tenta encontrar alternativas para `c1(R)`. Como quando o Prolog encontrou o corte ainda não tinha escolhido um valor para `R`, ainda pode tentar encontrar alternativas para ele. Quando acabam as alternativas para `R`, o Prolog já não pode procurar alternativas para `Q` e termina.

Para perceber como é que o corte altera o significado de um programa, consideremos o seguinte:

```
a2(X) :- b2(X), !, c2(X).
b2(1).
b2(2).
b2(3).
c2(2).
```

Sem o corte na primeira linha, o Prolog iria retornar `Q = 2` para o objectivo `a2(Q)`. Com o corte, o Prolog não consegue encontrar uma resposta. Para provar o objectivo, primeiro prova `b2(Q)` com `Q = 1`. A seguir encontra um corte, compromete-se com `Q = 1` e não consegue encontrar uma prova para `c2(1)`. Se pudesse ter procurado alternativas, teria encontrado `Q = 2`, que torna `b2(Q)` e `c2(Q)` ambos verdadeiros, mas o corte não o permite. No entanto, se perguntarmos directamente `a2(2)`, o Prolog responde `true`. Agora, o Prolog instancia o `X` da primeira linha com `2`, porque o utilizador o especificou, e quando chega ao corte está comprometido com o `2`, permitindo a prova de `c2(2)`.

O corte também funciona quando existem várias regras com a mesma cabeça. Se o programa tiver duas cláusulas para `a3(X)` e encontrar um corte numa dessas cláusulas, não só se compromete com as instanciações para as variáveis, mas também com a regra para `a3(X)`, não considerando as regras seguintes para este objectivo. Por exemplo, no programa seguinte:

```
a3(X) :- b3(X), !, c3(X).
a3(X) :- d3(X).
b3(1).
b3(4).
```

```
c3(3).
d3(4).
```

O Prolog falha com o objectivo `a3(X)`, apesar de poder resolver o objectivo com a segunda cláusula, usando `X = 4` e a última linha do programa, `d3(4)`.. Mas o Prolog tenta a primeira regra e quando encontra o corte tem que ignorar todas as alternativas para `a3(X)`. Neste caso, o objectivo `a3(4)` também falha, porque o Prolog também encontra o corte, uma vez que consegue provar `b3(4)`. De seguida, tenta provar `c3(4)` mas falha, mas como está comprometido com `X = 4` por causa do corte não pode procurar mais alternativas. Se a linha `b3(4)` fosse retirada do programa, o Prolog falharia na primeira regra, antes de encontrar o corte e poderia resolver o objectivo com a segunda regra.

Negação

O corte pode ser usado para implementar a negação como falhanço, em conjunto com o `fail`, que falha sempre.

Suponhamos que se usava o programa seguinte para responder ao objectivo `nao(G)`. A primeira regra é aplicada e tenta-se provar `G`. Se `G` tiver sucesso, o corte é encontrado, não são procuradas mais soluções, e é avaliado o `fail`, que falha sempre, logo `nao(G)` falha; se `G` falhar, não se continua na primeira cláusula (e por isso não se executa o corte) e executa-se a segunda cláusula, que tem sempre sucesso, e por isso `nao(G)` tem sucesso. Neste programa, a ordem das cláusulas é importante.

```
/*
  nao(X) :- nao se consegue provar X.
*/
nao(X) :- X, !, fail.
nao(_).
```

Neste programa, a terminação de `nao(G)` depende da terminação de `G`, isto é, se `G` não terminar, `nao(G)` pode ou não terminar, dependendo de ser ou não encontrada uma solução antes do ramo infinito.

Na realidade, o Prolog tem um predicado pré-definido `not/1`, que exprime uma forma de negação. Este predicado também pode ser usado em Prolog como um operador, usando `\+`, ou seja, `not(G)` é o mesmo que `\+ G`. Usando o predicado pré-definido, `not(G)` é verdadeiro se o Prolog não conseguir provar `G` como verdadeiro dada a base de dados actual. Quando o interpretador de Prolog encontra um objectivo da forma `not(G)`, tenta provar o objectivo `G`. Se o `G` conseguir ser provado, então `not(G)` falha, caso contrário, `not(G)` tem sucesso.

Existem programas que poderiam falhar usando a negação como falhanço, mas que não terminam usando as regras de computação do Prolog. Por exemplo, o objectivo `not((p1(X), p2(X)))` não termina relativamente ao programa seguinte, mas teria sucesso se `p2(X)` fosse selecionado primeiro, pois isso daria origem a uma árvore finitamente falhada.

```
p1(s(X)) :- p1(X).
```

```
p2(a).
p3(a).
p4(b).

%% ?- p1(X).
%% ERROR: Out of global stack
%% ?- p1(a).
%% false.
%%
%% ?- p2(X).
%% X = a.
%%
%% ?- not((p1(X), p2(X))).
%% ERROR: Out of global stack
%% ?- not((p2(X), p1(X))).
%% true.
%%
%% ?- not((p2(X), p3(X))).
%% false.
%%
%% ?- not((p2(X), p4(X))).
%% true.
%%
%% ?- not(p2(X)), not(p4(X)).
%% false.
%%
%% ?- p5(X).
%% ERROR: toplevel: Undefined procedure: p5/1 (DWIM could not correct goal)
%% ?- not(p2(X)), not(p5(X)).
%% ERROR: toplevel: Undefined procedure: p5/1 (DWIM could not correct goal)
%% ?- not((p2(X), p5(X))).
%% ERROR: toplevel: Undefined procedure: p5/1 (DWIM could not correct goal)
%% ?-
```

Outro exemplo é a relação de `estudanteSolteiro`, que define alguém que não é casado e é estudante:

```
/*
  estudanteSolteiro(X) :- tem sucesso quando X nao e casado e e estudante
*/
estudanteSolteiro(X) :- not(casado(X)), estudante(X).
estudante(rui).
casado(pedro).


%% ?- estudanteSolteiro(X).
%% false.
%%
%% ?- estudanteSolteiro(rui).
%% true.
```

```
%%
%% ?-
```

O objectivo `estudanteSolteiro(X)` falha com o programa anterior, apesar de `X=rui` ser uma solução implicada logicamente pelos factos conhecidos na base de conhecimento e de o objectivo `estudanteSolteiro(rui)` ter sucesso. O problema ocorre porque `not(casado(X))` falha, uma vez que `casado(X)` tem a solução `X=pedro`. Este problema pode ser evitado trocando a ordem das cláusulas e garantindo que as variáveis já estão instanciadas quando é usado o `not`.

Exercício 15.1
Escreva um programa que determina o mínimo entre dois números.

1. Usando Prolog puro.
2. Usando cortes.

Resposta:

1. ```
   /*
     minimo(X, Y, Min) :-  Min e o minimo dos numeros X e Y.
   */
   minimo(X, Y, X) :- X =< Y.
   minimo(X, Y, Y) :- X > Y.
   ```
2. ```
   /*
     minimo2(X, Y, Min) :- Min e o minimo dos numeros X e Y.
   */
   minimo2(X, Y, X) :- X =< Y, !.
   minimo2(X, Y, Y) :- X > Y, !.
   ```
   O corte usado neste programa é um corte verde, pois não altera as soluções existentes, uma vez que ambas as cláusulas têm testes que são mutuamente exclusivos. O corte na segunda cláusula aparece apenas para ficar mais parecida com a primeira, mas não é essencial.

Exercício 15.2
Explique porque é que o seguinte programa para o `minimo3` não tem os resultados esperados.

```
/*
  minimo3(X, Y, Min) :- Min e o minimo dos numeros X e Y.
*/
minimo3(X, Y, X) :- X =< Y, !.
minimo3(X, Y, Y).
```

Resposta:

Basta usar o objectivo `minimo3(2, 5, 5)`, que tem sucesso, apesar de `5` não ser o mínimo dos dois argumentos. O problema é que este programa foi feito pensando que o terceiro argumento não estaria unificado, e nesse caso funcionaria, mas no caso em que o terceiro argumento é conhecido podemos obter resultados errados. O programa correcto é o que estava na solução do exercício anterior.

Conclusão: nem sempre se podem eliminar condições que à partida poderiam parecer redundantes, pois o programa pode dar resultados errados se for chamado com outros argumentos que não os inicialmente esperados.

Exercício 15.3
Escreva um programa em Prolog que funde duas listas ordenadas de inteiros, tendo como resultado outra lista ordenada de inteiros, incluindo repetições. Por exemplo, `funde([1, 3, 5], [3, 7], [1, 3, 3, 5, 7])` tem o valor verdadeiro.

Resposta:

```
/*
  funde(Xs, Ys, Zs) :- Zs e uma lista ordenada de inteiros obtida
  atraves da fusao das listas ordenadas de inteiros Xs e Ys.
*/
funde([X|Xs], [Y|Ys], [X|Zs]) :-
   X < Y, !, funde(Xs, [Y|Ys], Zs).
funde([X|Xs], [Y|Ys], [X, Y|Zs]) :-
   X =:= Y, !, funde(Xs, Ys, Zs).
funde([X|Xs], [Y|Ys], [Y|Zs]) :-
   X > Y, !, funde([X|Xs], Ys, Zs).
funde(Xs, [], Xs) :- !.
funde([], Ys, Ys) :- !.

%% ?- funde([1, 3], [2], X).
%% X = [1, 2, 3].
%% ?- funde([1, 3, 5], [2, 3, 7], X).
%% X = [1, 2, 3, 3, 5, 7].
%% ?- funde(X, Y, [1, 2, 3, 3, 5, 7]).
%% ERROR: </2: Arguments are not sufficiently instantiated
%%    Exception: (6) funde(_G229, _G230, [1, 2, 3, 3, 5, 7]) ? a
%% % Execution Aborted
%% ?- funde([1, 3, 3], Y, [1, 2, 3, 3, 5, 7]).
%% ERROR: </2: Arguments are not sufficiently instantiated
%%    Exception: (7) funde([1, 3, 3], _G242, [1, 2, 3, 3, 5, 7]) ? a
%% % Execution Aborted
%% ?- funde(X, [1, 3, 3], [1, 2, 3, 3, 5, 7]).
%% ERROR: >/2: Arguments are not sufficiently instantiated
%%    Exception: (7) funde(_G241, [1, 3, 3], [1, 2, 3, 3, 5, 7]) ? a
%% % Execution Aborted
%% ?-
```

Neste programa, o corte da terceira cláusula não é necessário, mas está lá para tornar o programa mais "simétrico" e facilitar a sua leitura.

O seguinte programa, em que a primeira e a segunda cláusula se juntam, também funcionaria.

```
funde2([X|Xs], [Y|Ys], [X|Zs]) :-
        X =< Y, !, funde2(Xs, [Y|Ys], Zs).
funde2([X|Xs], [Y|Ys], [Y|Zs]) :-
        X > Y, !, funde2([X|Xs], Ys, Zs).
funde2(Xs, [], Xs) :- !.
funde2([], Ys, Ys) :- !.
```

Exercício 15.4
Em matemática, um polinómio é uma expressão construída a partir de uma ou mais variáveis e constantes, usando apenas os operadores de adição, subtracção e multiplicação, e expoentes inteiros positivos. Por exemplo, `x**2-4*x+7` é a representação de um polinómio em Prolog.

Considere o seguinte programa em Prolog para reconhecer polinómios.

```
/*
  polinomio(Termo, X) :- Termo e um polinomio em X.
```

```
*/
polinomio(X, X).
polinomio(Termo, _) :-
   number(Termo).
polinomio(Termo1+Termo2, X) :-
   polinomio(Termo1, X), polinomio(Termo2, X).
polinomio(Termo1-Termo2, X) :-
   polinomio(Termo1, X), polinomio(Termo2, X).
polinomio(Termo1*Termo2, X) :-
   polinomio(Termo1, X), polinomio(Termo2, X).
polinomio(Termo1/Termo2, X) :-
   polinomio(Termo1, X), number(Termo2).
polinomio(Termo**N, X) :-
   integer(N), N >= 0, polinomio(Termo, X).
```

Altere-o, usando cortes verdes, de modo a eliminar ramos desnecessários da árvore de refutação.

Resposta:

```
/*
  polinomio2(Termo, X) :- Termo e um polinomio em X.
  Usa cortes verdes.
*/
polinomio2(X, X) :- !.
polinomio2(Termo, _) :-
   number(Termo), !.
polinomio2(Termo1+Termo2, X) :-
   !, polinomio2(Termo1, X), polinomio2(Termo2, X).
polinomio2(Termo1-Termo2, X) :-
   !, polinomio2(Termo1, X), polinomio2(Termo2, X).
polinomio2(Termo1*Termo2, X) :-
   !, polinomio2(Termo1, X), polinomio2(Termo2, X).
polinomio2(Termo1/Termo2, X) :-
   !, polinomio2(Termo1, X), number(Termo2).
polinomio2(Termo**N, X) :-
   !, integer(N), N >= 0, polinomio2(Termo, X).

%% ?- polinomio2(4, x).
%% true.
%% ?- polinomio2(4*xy, xy).
%% true.
%% ?- polinomio2(4*xy, x).
%% false.
%% ?- polinomio2(4*xy+2*xy**4, xy).
%% true.
%% ?-
```

Exercício 15.5
Explique qual é o problema com o corte no seguinte programa:

```
/*
```

```
  membro3(X, L) :- X e um membro de L.
*/
membro3(X, [X|_]) :- !.
membro3(X, [_|Ys]) :- membro3(X, Ys).
```

Resposta:

O problema é que este é um corte vermelho, que altera o significado do programa. Quando quisermos usar o predicado `membro3` para saber quais são os elementos de uma lista, apenas vamos conseguir obter o primeiro, pois o corte elimina a possibilidade de fazer retrocesso e encontrar os outros elementos.

A interacção seguinte ilustra alguns dos problemas:

```
%% ?- membro3(2, [1, 2, 3]).
%% true.
%% ?- membro3(X, [1, 2, 3]).
%% X = 1.
%% ?- membro3(X, [1, 2, 3]), X=2.
%% false.
%% ?- X=2, membro3(X, [1, 2, 3]).
%% X = 2.
%% ?-
```

O primeiro objectivo dá o resultado esperado.

O segundo objectivo apenas tem uma solução que corresponde ao primeiro elemento da lista, quando deveria ter 3 soluções.

No terceiro objectivo a resposta é `false`, quando existe a solução `X = 2`.

No quarto objectivo, conseguimos ter essa solução porque quando chamamos o `membro3` o `X` já está unificado com `2` e só na chamada recursiva ao `membro3` é que vai unificar com o segundo elemento da lista.

Exercício 15.6
Escreva um programa `separa(Numeros, Positivos, Negativos)`, que separa uma lista de números em duas listas, uma com os números positivos e outra com os números negativos. Considere que o zero é um número positivo. Por exemplo, `separa([1, -2, 0, -3], [1, 0], [-2, -3])` tem o valor verdadeiro.

Resposta:

```
/*
  separa(Numeros, Positivos, Negativos) :- Positivos contem os numeros
  positivos da lista Numeros e Negativos contem os numeros negativos
  da lista Numeros.
*/
separa([N | Resto], [N | Positivos], Negativos) :-
       N >= 0, !, separa(Resto, Positivos, Negativos).
separa([N | Resto], Positivos, [N | Negativos]) :-
       N < 0, !, separa(Resto, Positivos, Negativos).
separa([], [], []).
```

```
%% ?- separa([1, -2, 5, -3], P, N).
%% P = [1, 5],
%% N = [-2, -3].
%% ?-  separa([1, -2, 5, -3], [1], N).
%% false.
%% ?- separa([1, -2, 5, -3], [], N).
%% false.
%% ?-  separa([1, -2, 5, -3], [1, 5], N).
%% N = [-2, -3].
%% ?- separa([1, -2, 5, -3], P, []).
%% false.
%% ?- separa([1, -2, 5, -3], P, [-2]).
%% false.
%% ?- separa([1, -2, 5, -3], P, [-2, -3]).
%% P = [1, 5].
%% ?- separa(X, [0, 3, 4], [-2, -1]).
%% X = [0, 3, 4, -2, -1] ;
%% false.
%% ?- separa(X, P, [-2, -1]).
%% ERROR: >=/2: Arguments are not sufficiently instantiated
%%    Exception: (7) separa(_G220, _G221, [-2, -1]) ? a
%% % Execution Aborted
%% ?- separa(X, [0, 3, 4], N).
%% X = [0, 3, 4],
%% N = [] ;
%% ERROR: </2: Arguments are not sufficiently instantiated
%% ?-
```

Exercício 15.7
Escreva um programa `ifThenElse(A, B, C)`, que caso `A` tenha sucesso avalia `B` e caso `A` falhe avalia `C`.

Resposta:

```
/*
  ifThenElse(A, B, C) :- caso A tenha sucesso avalia B
  e caso A falhe avalia C.
*/
ifThenElse(A, B, _) :- A, !, B.
ifThenElse(_, _, C) :- C.

%% ?- ifThenElse(2=:=1+1, writeln('avaliei B'), writeln('avaliei C')).
%% avaliei B
%% true.
%%
%% ?- ifThenElse(2==1+1, writeln('avaliei B'), writeln('avaliei C')).
%% avaliei C
%% true.
%%
%% ?- ifThenElse(2=:=1+1, 2==1+1, writeln('avaliei C')).
%% false.
%%
%% ?-
```

Convém notar que neste programa a ordem das cláusulas é relevante.

Uma alternativa, usando `;`, seria a seguinte:

```
/*
  ifThenElse2(A, B, C) :- caso A tenha sucesso avalia B
  e caso A falhe avalia C.
*/
ifThenElse2(A, B, C) :- A, !, B; C.

%% ?- ifThenElse2(2=:=1+1, writeln('avaliei B'), writeln('avaliei C')).
%% avaliei B
%% true.
%%
%% ?- ifThenElse2(2==1+1, writeln('avaliei B'), writeln('avaliei C')).
%% avaliei C
%% true.
%%
%% ?- ifThenElse2(2=:=1+1, (writeln('avaliei B'), fail), writeln('avaliei C')).
%% avaliei B
%% false.
%%
%% ?- ifThenElse2(2=:=1+1, 2==1+1, writeln('avaliei C')).
%% false.
%%
%% ?-
```

Exercício 15.8
Considere o programa:

```
m(1).
m(2) :- !.
m(3).
```

Diga quais são todas as respostas do Prolog aos seguintes objectivos, considerando que o utilizador escreve `;` até esgotar todas as respostas:

1. `?- m(X).`
2. `?- m(X), m(Y).`
3. `?- m(X), !, m(Y).`
4. `?- m(X), m(Y), !.`
5. `?- m(1), m(2), m(3).`

Resposta:

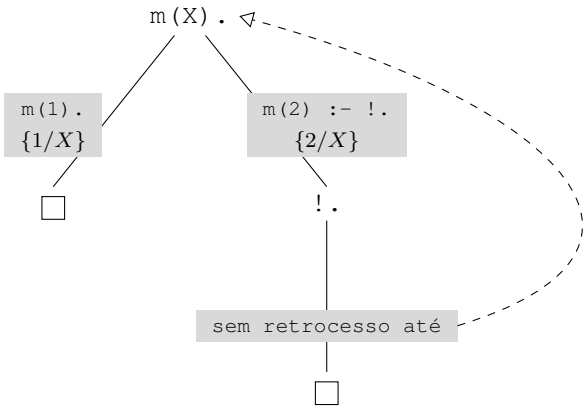
Interacção com o Prolog:

```
%% ?- m(X).
%% X = 1 ;
%% X = 2.
%% ?- m(X), m(Y).
```

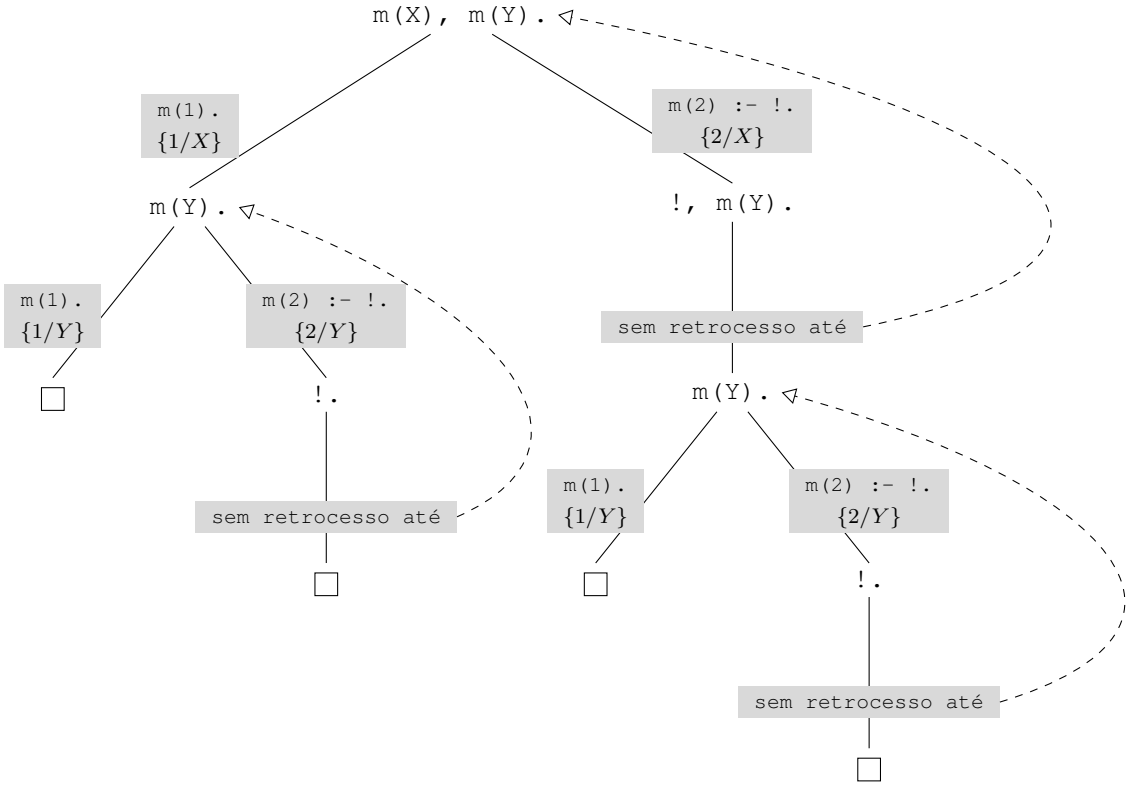
```
%% X = Y, Y = 1 ;
%% X = 1,
%% Y = 2 ;
%% X = 2,
%% Y = 1 ;
%% X = Y, Y = 2.
%% ?- m(X), !, m(Y).
%% X = Y, Y = 1 ;
%% X = 1,
%% Y = 2.
%% ?-  m(X), m(Y), !.
%% X = Y, Y = 1.
%% ?- m(1), m(2), m(3).
%% true.
%% ?-
```

Árvores SLD:

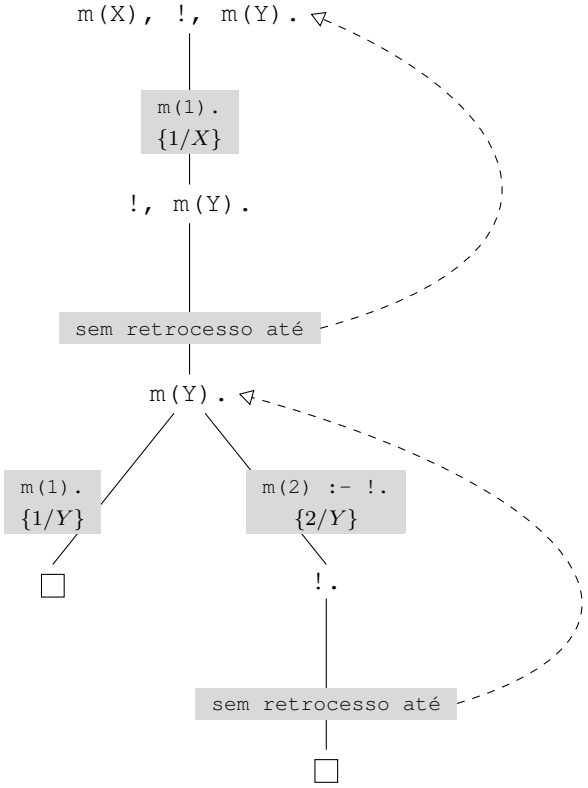
1. `?- m(X).`



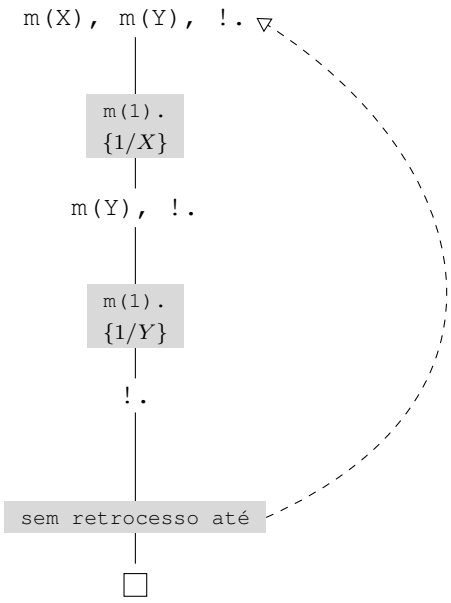
2. `?- m(X), m(Y).`


m(X), m(Y).
m(1).
{1/X}
m(2) :- !.
{2/X}
m(Y).
!, m(Y).
m(1).
{1/Y}
m(2) :- !.
{2/Y}
sem retrocesso até
!.
sem retrocesso até
m(Y).
m(1).
{1/Y}
m(2) :- !.
{2/Y}
!.
sem retrocesso até

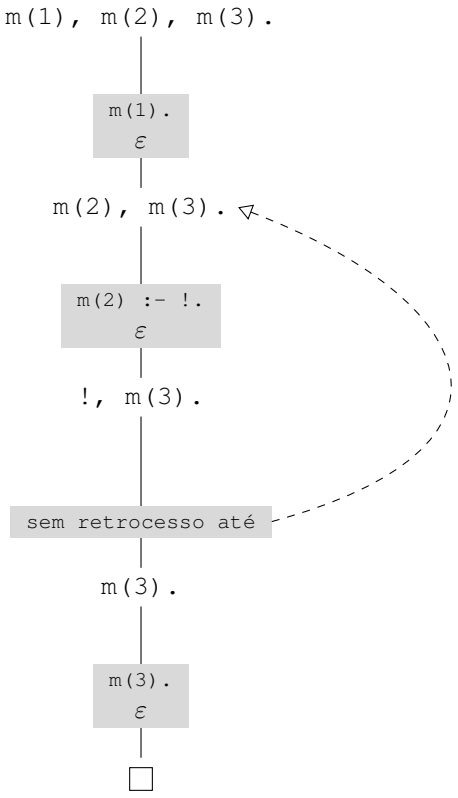
3. `?- m(X), !, m(Y).`


m(X), !, m(Y).
m(1).
{1/X}
!, m(Y).
sem retrocesso até
m(Y).
m(1).
{1/Y}
m(2) :- !.
{2/Y}
!.
sem retrocesso até

4. `?- m(X), m(Y), !.`



5. `?- m(1), m(2), m(3).`



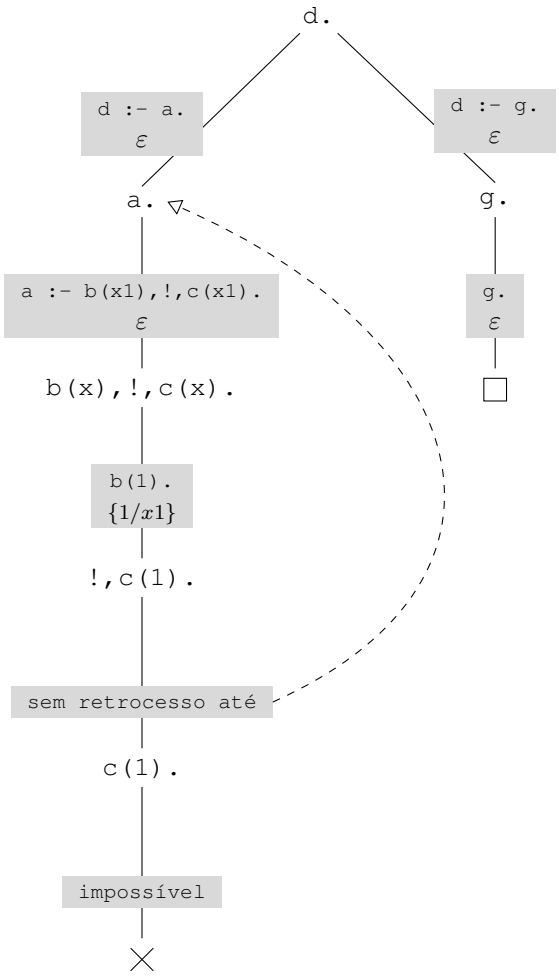
Exercício 15.9
Considere o seguinte programa em Prolog.

```
d :- a.
d :- g.
a :- b(X), !, c(X).
a :- e.
b(1).
```

```
b(2).
c(2).
g.
e.
```

Construa a árvore de refutação para mostrar que `d` sucede através de `g`.

Resposta:



Exercício 15.10
Considere o seguinte programa:

```
p(X, Y) :- q(X, Y).
p(a, b).
q(c, d).
q(e, f).
q(X, Y) :- r(X), !, s(Y).
q(X, Y) :- t(X, Y).
r(e).
r(f).
s(g).
s(h).
```

```
t(i, j).
```

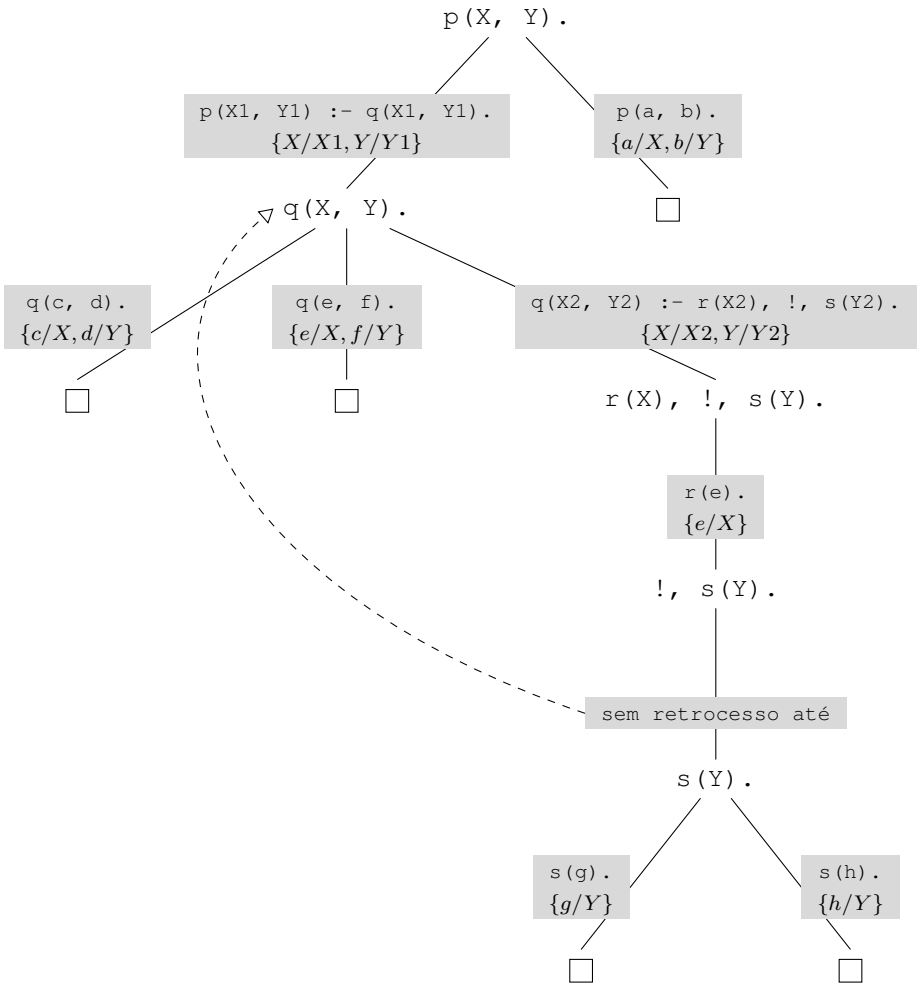
Diga quais são as respostas dadas pelo Prolog ao objectivo `p(X, Y)`, considerando que o utilizador escreve `;` até esgotar todas as respostas.

Resposta:

Interacção com o Prolog:

```
?- p(X, Y).
X = c,
Y = d ;
X = e,
Y = f ;
X = e,
Y = g ;
X = e,
Y = h ;
X = a,
Y = b.

?-
```

Árvore SLD:



Exercício 15.11
Considere o seguinte programa:

```
u(a).
u(b).

v(1).
v(2).
v(3).

w1(X, Y) :- !, u(X), v(Y).
w2(X, Y) :- u(X), !, v(Y).
w3(X, Y) :- u(X), v(Y), !.
```

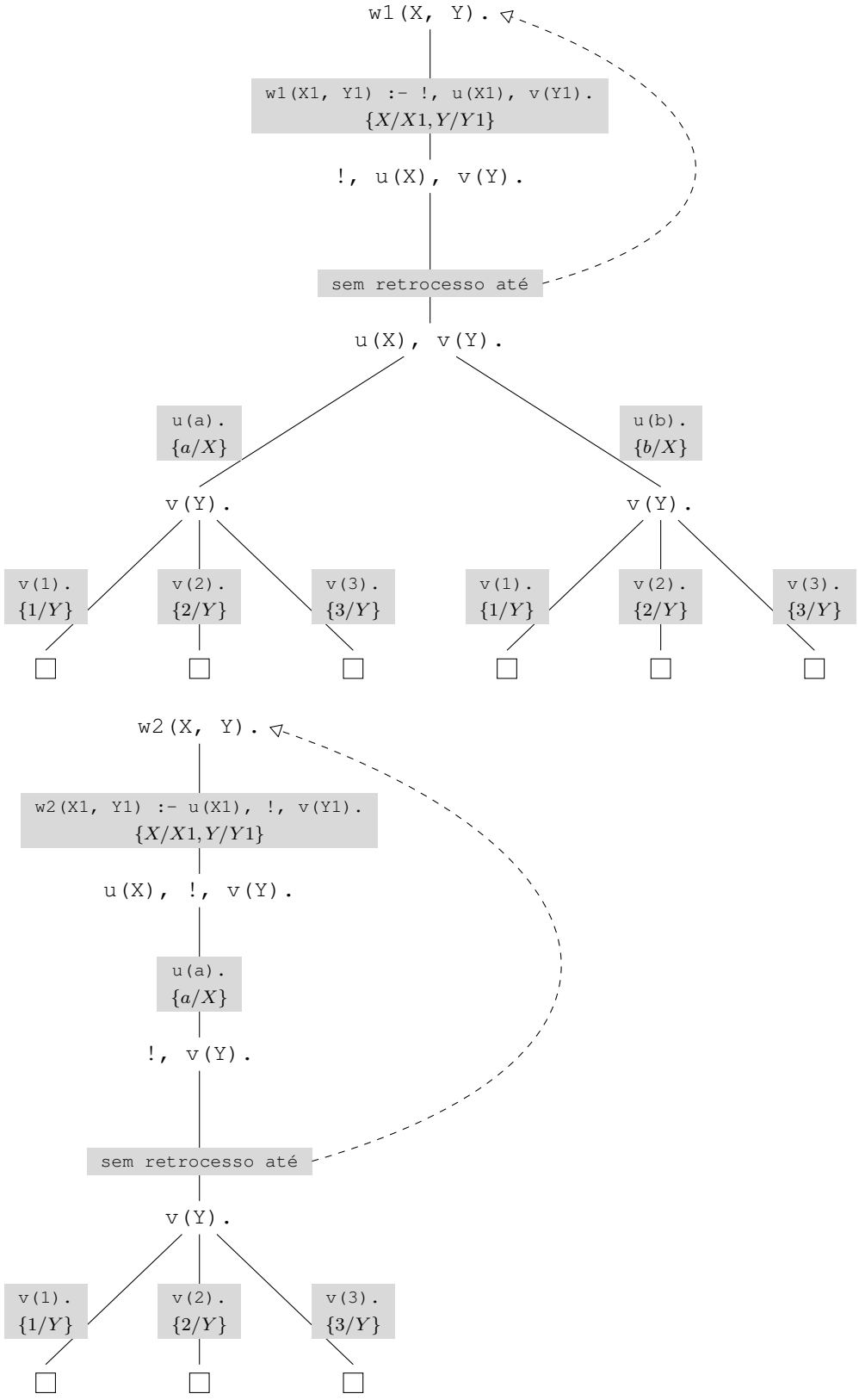
Diga quais são as respostas dadas pelo Prolog aos objectivos `w1(X, Y)`, `w2(X, Y)`, e `w3(X, Y)`, considerando que o utilizador escreve `;` até esgotar todas as respostas.

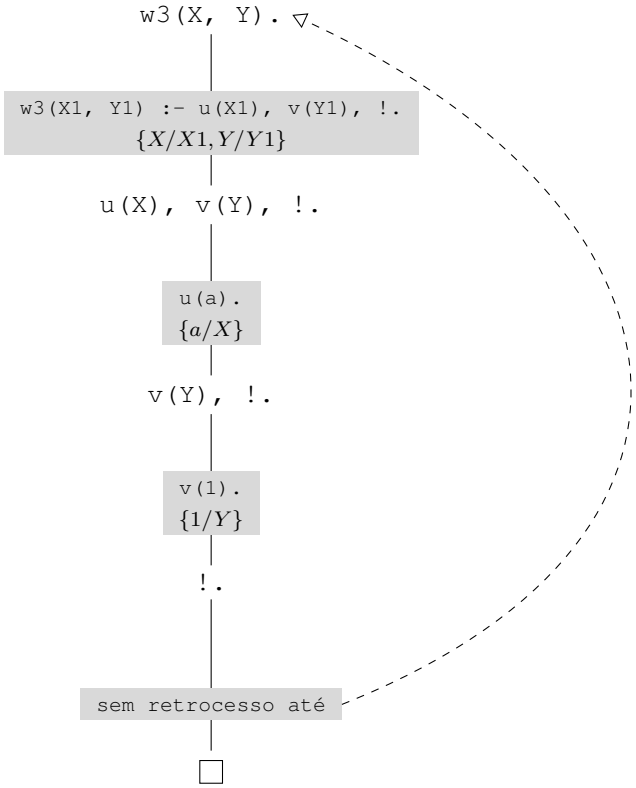
Resposta:

Interacção com o Prolog:

```
?- w1(X, Y).
X = a,
Y = 1 ;
X = a,
Y = 2 ;
X = a,
Y = 3 ;
X = b,
Y = 1 ;
X = b,
Y = 2 ;
X = b,
Y = 3.

?- w2(X, Y).
X = a,
Y = 1 ;
X = a,
Y = 2 ;
X = a,
Y = 3.

?- w3(X, Y).
X = a,
Y = 1.

?-
```

Árvores SLD:


w1(X, Y).
w1(X1, Y1) :- !, u(X1), v(Y1).
{X/X1, Y/Y1}
!, u(X), v(Y).
sem retrocesso até
u(X), v(Y).
u(a).
{a/X}
u(b).
{b/X}
v(Y).
v(Y).
v(1).
{1/Y}
v(2).
{2/Y}
v(3).
{3/Y}
v(1).
{1/Y}
v(2).
{2/Y}
v(3).
{3/Y}
w2(X, Y).
w2(X1, Y1) :- u(X1), !, v(Y1).
{X/X1, Y/Y1}
u(X), !, v(Y).
u(a).
{a/X}
!, v(Y).
sem retrocesso até
v(Y).
v(1).
{1/Y}
v(2).
{2/Y}
v(3).
{3/Y}



Exercício 15.12
Escreva um predicado de dois argumentos `diferentes`, que tem sucesso apenas quando os seus dois argumentos não são o mesmo, isto é, não são unificáveis.

Resposta:

```
/*
  diferentes(X, Y) :- tem sucesso quando X e Y sao diferentes.
*/
diferentes(X, X) :- !, fail.
diferentes(_, _).
```

Usando o `not`, pode ficar:

```
diferentes2(X, Y) :- not(X = Y).
```

Usando o diferente do Prolog, `\=`, pode ficar:

```
diferentes3(X, Y) :- X \= Y.
```

Exemplos de utilização:

```
%% ?- diferentes(a, b).
%% true.
%% ?- diferentes(A, B).
%% false.
```

```
%% ?- diferentes(1, 1).
%% false.
%% ?- diferentes(1, 2-1).
%% true.
%% ?- 1=2-1.
%% false.
%% ?- 1=:=2-1.
%% true.
%% ?- diferentes2(a, b).
%% true.
%% ?- diferentes2(A, B).
%% false.
%% ?- diferentes2(1, 1).
%% false.
%% ?- diferentes2(1, 2-1).
%% true.
%% ?- diferentes3(a, b).
%% true.
%% ?- diferentes3(A, B).
%% false.
%% ?- diferentes3(1, 1).
%% false.
%% ?- diferentes3(1, 2-1).
%% true.
%% ?-
```

Exercício 15.13
Defina o conceito de duas listas serem disjuntas usando o `not`, partindo do princípio que existe o predicado `membro/2` que indica se um elemento é membro de uma lista .

Resposta:

```
/*
  disjuntas(X, Y) :- tem sucesso quando X e Y sao listas disjuntas.
*/
disjuntas(Xs, Ys) :- not((membro(Z, Xs), membro(Z, Ys))).

%% ?- disjuntas([a, b, c], [1, 2, 3]).
%% true.
%% ?- disjuntas([1, 2, c], [a, 2, 4]).
%% false.
%% ?-
```

Exercício 15.14
Considere o seguinte programa em Prolog.

```
pessoaAlta(X) :- not(baixa(X)), pessoa(X).
pessoa(eva).
pessoa(maria).
baixa(maria).
```

Qual a resposta do Prolog ao objectivo `pessoaAlta(X)`? Corresponde ao que estaria intuitivamente correcto? Se não, explique como é que poderia passar a corresponder.

Resposta:

A resposta do Prolog é `false`, mas estariamos à espera que fosse `X = eva`, até porque o objectivo `pessoaAlta(eva)` tem sucesso. Para obtermos esta resposta, temos que garantir que o `X` já está instanciado quando é avaliado o `not`, o que podemos fazer trocando a ordem dos literais na cláusula:

```
pessoaAlta1(X) :- pessoa(X), not(baixa(X)).
```

Exemplos de utilização:

```
%% ?- pessoaAlta(X).
%% false.
%% ?- pessoaAlta(eva).
%% true.
%% ?- pessoaAlta(maria).
%% false.
%% ?- pessoaAlta1(X).
%% X = eva ;
%% false.
%% ?- pessoaAlta1(eva).
%% true.
%% ?- pessoaAlta1(maria).
%% false.
%% ?-
```

Exercício 15.15
Considere o seguinte programa:

```
p1(s(X)) :- p1(X).
p2(a).
```

Diga qual a resposta do Prolog ao objectivo `not((p1(X), p2(X)))`.

Resposta:

```
?- not((p1(X), p2(X))).
ERROR: Out of global stack
?-
```

Para este objectivo, o Prolog não termina, mas teria sucesso se `p2(X)` fosse selecionado primeiro, por exemplo em `not((p2(X), p1(X)))`, pois isso daria origem a uma árvore finitamente falhada.

```
?- not((p2(X), p1(X))).
true.
?-
```

Exercício 15.16
Considere a seguinte base de conhecimento:

```
cao(bobi).
cao(fiel).
cao(guerreiro).
morde(guerreiro).
```

E as duas formas de representar que o Carlos gosta de todos os cães que não mordam. Repare que em termos lógicos não existem diferenças entre as duas.

```
gosta1(carlos, X) :- cao(X), not(morde(X)).

gosta2(carlos, X) :- not(morde(X)), cao(X).
```

Diga qual a resposta do Prolog a cada um dos seguintes objectivos `gosta1(carlos, X)` e `gosta2(carlos, X)`. Se as respostas forem diferentes, explique a razão dessas diferenças.

Resposta:

```
%% ?- gosta1(carlos, X).
%% X = bobi ;
%% X = fiel ;
%% false.
%%
%% ?- gosta2(carlos, X).
%% false.
%%
%% ?- halt.
%%
%% Process prolog finished
```

No primeiro caso, primeiro o Prolog vai tentar encontrar os cães e só depois de estarem instanciados é que vai tentar provar que não mordem. Neste caso, os resultados são os esperados.

No segundo caso, a variável `X` não está instanciada quando o Prolog vai tentar provar `not(morde(X))`, e este objectivo falha porque existe algo que morde na base de conhecimento.

Convém notar que ambos os objectivos dariam origem a um erro se não existisse nada que mordesse nem nenhuma definição para o que significa morder na base de conhecimento, pois o predicado não estaria definido.