

Capítulo 1

Fundamentos da lógica matemática

1.1 Introdução

A resolução é uma regra de inferência apropriada para dedução automática mecânica. Foi precisamente no contexto de dedução automática que Robinson desenvolveu o seu princípio de resolução, publicado em 1965. A idéia de que a lógica de primeira-ordem poderia ser usada como uma linguagem de programação foi revolucionária, já que até 1972 a lógica tinha sido considerada apenas como uma linguagem de especificação ou declaração. Kowalski mostra que a lógica possui uma interpretação procedimental o que a faz efetiva como uma linguagem de programação. Em resumo, uma *cláusula de programa* $A \leftarrow B_1, \dots, B_n$ é observada como uma definição de procedimento. Se $\leftarrow C_1, \dots, C_k$ é um “objetivo” então cada C_j ($1 \leq j \leq k$) é considerado como um chamado a um procedimento. Um programa é executado apresentando um objetivo inicial. Se o objetivo num momento da computação é

$$\leftarrow C_1, \dots, C_k \tag{1.1}$$

um passo na execução consiste numa unificação de algum C_j ($1 \leq j \leq k$) com a *cabeça* de uma cláusula de programa $A \leftarrow B_1, \dots, B_n$. Assim, o objetivo (1.1) é reduzido a

$$(\leftarrow C_1, \dots, C_{j-1}, B_1, \dots, B_n, C_{j+1}, \dots, C_k)\theta,$$

onde θ é a substituição da unificação de A e C_j . A execução termina quando o objetivo vazio é atingido.

A idéia fundamental de Kowalski é que um algoritmo consiste de dois componentes distintos:

- lógica e
- controle

O componente lógico estabelece o **que** o programa deve resolver e o componente de controle **como** deve resolvê-lo. Em termos gerais, uma linguagem lógica de programação deveria prover mecanismos para especificar cada um dos componentes. Não obstante, considera-se ideal liberar o programador da especificação do componente de controle; fato este que ainda não é possível com os

atuais sistemas de programação lógica, e que seguramente só será atingido sob restrições no tipo de declarações lógicas. Inicialmente estudamos a sintaxe e semântica das teorias lógicas de primeira-ordem, o mecanismo de unificação (capítulo 2) e teoria do ponto fixo (capítulo 3) dentro do contexto da lógica matemática. Tais tópicos representam os fundamentos necessários para o estudo formal do princípio de resolução e da programação declarativa.

1.2 Teorias de primeira-ordem

Iniciaremos com os fundamentos da lógica de primeira-ordem. A seguinte bibliografia complementar é recomendada: [Cai88], [Fit96], [Ber95], [End72], [CK90], [Gal87], [Bur98] e [BE93].

A lógica de primeira-ordem deve ser estudada tanto do ponto de vista sintático (o **como** escrever as fórmulas e predicados) como do ponto de vista semântico (o **que** significam as fórmulas e predicados). Iniciaremos com os aspectos puramente sintáticos da lógica de primeira-ordem.

Uma **teoria de primeira-ordem** consiste de um **alfabeto**, uma **linguagem de primeira-ordem**, um conjunto de **axiomas** e um conjunto de **regras de inferência**. A linguagem de primeira-ordem consiste de fórmulas **bem formadas** da teoria. Os axiomas são um conjunto de fórmulas bem formadas. Os axiomas e regras de inferência são usados para derivar os teoremas da teoria.

Definição 1.2.1 *Um alfabeto consiste de sete classes de símbolos:*

- (a) *variáveis,*
- (b) *constantes,*
- (c) *símbolos de função,*
- (d) *símbolos de predicado,*
- (e) *conectivos (lógicos),*
- (f) *quantificadores (lógicos),*
- (g) *símbolos de pontuação.*

As classes (e), (f) e (g) são sempre as mesmas para qualquer alfabeto, enquanto que as classes (a), (b), (c) e (d) variam segundo o alfabeto (e a teoria que especificam).

Notação: Utilizaremos os seguintes conjuntos de letras para denotar:

- Variáveis: u, v, w, x, y e z .
- Constantes: a, b e c .

- Símbolos de funções: f, g e h .
- Símbolos de predicados: p, q e r .

As funções serão de um ou mais argumentos (funções de zero argumentos são constantes), enquanto os predicados terão zero ou mais argumentos. •

Definição 1.2.2 *O número de argumentos de um símbolo de função (ou predicado) é denominado a “aridade” (arity) da função (ou do predicado). Um símbolo de função ou predicado de aridade n é denominado um símbolo n -ário.*

Os conectivos lógicos são $\neg, \wedge, \vee, \rightarrow$ e \leftrightarrow , enquanto os quantificadores são \exists e \forall . Os símbolos de pontuação são “(”, “)” e “,”.

A semântica informal dos conectivos e quantificadores é a seguinte:

- \neg é a negação (“não”),
- \wedge é a conjunção (“e”),
- \vee é a disjunção (“ou”),
- \rightarrow é a implicação (“se ... então”),
- \leftrightarrow é a equivalência (“se e somente se”),
- “ $\forall x$ ” significa “para todo x ” e,
- “ $\exists x$ ” significa “existe x ”.

Definição 1.2.3 *Um termo é definido indutivamente como segue:*

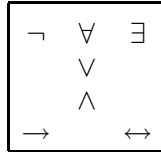
- Uma variável é um termo.
- Uma constante é um termo.
- Se f é um símbolo de função de aridade n e t_1, \dots, t_n são termos então $f(t_1, \dots, t_n)$ é um termo.

Definição 1.2.4 *Uma fórmula (bem formada) é definida indutivamente como segue:*

- Se p é um símbolo de predicado de aridade n e t_1, \dots, t_n são termos, então $p(t_1, \dots, t_n)$ é uma fórmula, normalmente denominada **fórmula atômica** ou simplesmente **átomo**.
- Se F e G são fórmulas, então $(\neg F)$, $(F \wedge G)$, $(F \vee G)$, $(F \rightarrow G)$, $(F \leftrightarrow G)$ são fórmulas

- Se F é uma fórmula e x uma variável, então $(\forall_x F)$ e $(\exists_x F)$ são fórmulas.

Para evitar o excesso de parênteses adota-se a seguinte ordem de precedência:



Exemplo 1.2.5 A fórmula $p(x) \vee q(x) \wedge r(x)$ deve-se interpretar como $(p(x) \vee q(x)) \wedge r(x)$. ◇

Por conveniência, as fórmulas $F \rightarrow G$ serão escritas na forma $(G \leftarrow F)$.

Definição 1.2.6 A linguagem de primeira-ordem dada por um alfabeto, consiste do conjunto de fórmulas construídas com os símbolos do alfabeto.

Exemplo 1.2.7 As seguintes expressões

a) $(\forall_x(\neg(\exists_z(\neg(P(x, y) \rightarrow r(z)))))$ e

b) $((\forall_x P(x)) \vee (\neg P(x)))$

são fórmulas construídas utilizando-se a linguagem de primeira-ordem de um alfabeto que inclui os símbolos usados nas próprias fórmulas.

Usando a ordem de precedência definida anteriormente, podemos simplificar essas fórmulas, como segue:

a') $\forall_x \neg \exists_z \neg (P(x, y) \rightarrow r(z))$ e

b') $\forall_x P(x) \vee \neg P(x)$, respectivamente. ◇

Definição 1.2.8 A abrangência de \forall_x (respectivamente, \exists_x) em $\forall_x F$ (respectivamente, $\exists_x F$) é F . Uma ocorrência de uma **variável ligada** numa fórmula G , é uma ocorrência de uma variável x , dentro do campo de abrangência de um quantificador \forall_x ou \exists_x . Uma ocorrência de uma **variável livre** é uma ocorrência de uma variável x não ligada.

Exemplo 1.2.9 Na fórmula $\exists_x(p(f(x), y) \rightarrow q(x))$, as duas ocorrências da variável x são ligadas, enquanto a ocorrência da variável y é livre. Na fórmula $\exists_x p(f(x), y) \rightarrow q(x)$ a primeira ocorrência da variável x é ligada, no entanto a segunda é livre. ◇

Definição 1.2.10 Uma **fórmula fechada** ou uma **sentença** é uma fórmula sem ocorrências livres de nenhuma variável.

Notação: Normalmente, falaremos de *variáveis livres* (ou *ligadas*) quando todas as ocorrências da variável em questão são livres (ou ligadas, respectivamente). •

Exemplo 1.2.11 Na fórmula $\exists x(p(f(x), y) \rightarrow q(x))$, x é uma variável ligada e y uma variável livre. Na fórmula $\exists x p(x, y) \rightarrow q(x)$ não podemos dizer que x seja uma variável livre ou ligada. ◇

☞ Uma fórmula sem variáveis livres corresponde a uma sentença.

Exemplo 1.2.12 $\exists x \forall y (p(f(x), h(x, y)) \rightarrow q(y))$ é uma fórmula fechada ou sentença. ◇

Definição 1.2.13 Se F é uma fórmula, $\forall(F)$ denota seu **fecho universal** que corresponde à fórmula obtida de F , precedida de um quantificador universal para cada variável com ocorrências livres em F . Analogamente é definido o **fecho existencial** de F , $\exists(F)$.

Exemplo 1.2.14 O fecho universal da fórmula $\forall x p(f(x, y), h(z)) \rightarrow q(x)$ é:

$$\forall y \forall z \forall x (\forall x p(f(x, y), h(z)) \rightarrow q(x)).$$

O correspondente fecho existencial é:

$$\exists y \exists z \exists x (\forall x p(f(x, y), h(z)) \rightarrow q(x)).$$

Note que existe uma ocorrência livre e uma ligada da variável x na fórmula original. ◇

Definição 1.2.15 Definimos indutivamente o tipo de ocorrência de um átomo numa fórmula da seguinte maneira:

- Um átomo A **ocorre positivamente** em A .
- Se um átomo A ocorre positivamente (resp. negativamente) numa fórmula W , então A **ocorre positivamente**, (resp. **negativamente**) em $\exists x W$ e $\forall x W$ e $W \wedge V$ e $W \vee V$ e $W \leftarrow V$ e **ocorre negativamente** (resp. **postivamente**) em $\neg W$ e $V \leftarrow W$.

Observação: $F \leftarrow G$ é equivalente¹ a $\neg G \vee F$. •

¹A palavra “equivalente” está enfatizada, porque que ainda não definimos formalmente o que isto significa (ver Definição 1.3.31).

Definição 1.2.16 Um **literal** é um átomo ou a negação de um átomo. Distingüe-se entre **literal positivo** e **literal negativo** segundo o literal seja um átomo ou a negação de um átomo, respectivamente.

Definição 1.2.17 Uma **cláusula** é uma fórmula da forma:

$$\boxed{\forall x_1 \dots \forall x_s (L_1 \vee \dots \vee L_m),}$$

onde cada L_i é um literal e x_1, \dots, x_s são todas as variáveis ocorrendo em $L_1 \vee \dots \vee L_m$.

Exemplo 1.2.18 As seguintes fórmulas são cláusulas:

$$\forall x \forall y \forall z (p(x, y) \vee \neg p(x, z) \vee r(x));$$

$$\forall x \forall y \forall z (p(f(x, y), y) \vee r(h(z))).$$

◇

Notação: Como as cláusulas são de uso comum na programação lógica, existe uma notação padrão. Denotaremos uma cláusula da forma:

$$\boxed{\forall (A_1 \vee \dots \vee A_r \vee \neg B_1 \vee \dots \vee \neg B_n)}$$

onde os A_i 's são os literais positivos e os $\neg B_j$'s os literais negativos (da cláusula) por:

$$\boxed{A_1, \dots, A_r \leftarrow B_1, \dots, B_n}$$

assumindo que todas as variáveis estão quantificadas universalmente. •

Observações:

1. Note que $\forall (A_1 \vee \dots \vee A_r \vee \neg B_1 \vee \dots \vee \neg B_n)$ e $\forall (A_1 \vee \dots \vee A_r \leftarrow B_1 \wedge \dots \wedge B_n)$ são *equivalentes*. Com efeito, $\forall (A_1 \vee \dots \vee A_r \vee \neg B_1 \vee \dots \vee \neg B_n)$ é *equivalente* à $\forall ((A_1 \vee \dots \vee A_r) \vee \neg (B_1 \wedge \dots \wedge B_n))$ pelas leis de DeMorgan da lógica proposicional.
2. O símbolo “ \equiv ” denota a *equivalência* de fórmulas que será definida formalmente na Seção 1.3 (ver Definição 1.3.31). Assim,

$$\forall ((A_1 \vee \dots \vee A_r) \vee \neg (B_1 \wedge \dots \wedge B_n)) \equiv$$

$$\forall (A_1 \vee \dots \vee A_r \leftarrow B_1 \wedge \dots \wedge B_n).$$

•

Definição 1.2.19 *Uma cláusula de definição de programa é uma cláusula com um único literal positivo.*

$$\boxed{A \leftarrow B_1, \dots, B_n}$$

A é denominado a **cabeça** e B_1, \dots, B_n o **corpo** da cláusula.

Definição 1.2.20 *Uma cláusula unitária é uma cláusula da forma*

$$\boxed{A \leftarrow}$$

i.e., uma cláusula de definição de programa com corpo vazio.

Definição 1.2.21 *Um programa definido (ou simplesmente um programa) é um conjunto finito de cláusulas de definição de programa.*

Definição 1.2.22 *Em um programa P todas as cláusulas de definição com o mesmo símbolo de predicado p , na cabeça, é denominado o conjunto de definição de p .*

A interpretação informal de uma cláusula de definição de programa

$$\boxed{A \leftarrow B_1, \dots, B_n}$$

é: “Toda **instância** de A com correspondentes instâncias de B_1, \dots, B_n válidas, é válida”. Uma instância de A resulta da *substituição*² das suas variáveis.

Definição 1.2.23 *Um objetivo (definite goal) é uma cláusula da forma:*

$$\boxed{\leftarrow B_1, \dots, B_n}$$

Cada B_j é denominado um **sub-objetivo** do objetivo.

Se y_1, \dots, y_r são as variáveis que ocorrem no objetivo, então sua notação clausal será: $\forall y_1, \dots, \forall y_r (\neg(B_1 \wedge \dots \wedge B_n))$ que, por sua vez, é equivalente à $\neg \exists y_1 \dots \exists y_r (B_1 \wedge \dots \wedge B_n)$.

Definição 1.2.24 *A cláusula vazia sem antecedente, nem consequente é denotada por \square .*

Definição 1.2.25 *Uma cláusula de Horn é uma cláusula de definição de programa ou um objetivo.*

²O termo “substituição” é enfatizado, já que ainda não temos definido formalmente o que isto significa (ver Definição 2.1.1).

Exemplo 1.2.26 [Llo87] O seguinte programa denominado *slowsort* (ordenação lenta), ordena uma lista de inteiros não negativos em uma lista crescente (a lista original não inclui repetições).

$$\begin{array}{ll}
\text{sort}(x, y) & \leftarrow \text{sorted}(y), \text{perm}(x, y) \\
\text{sorted}(\text{nil}) & \leftarrow \\
\text{sorted}(x \cdot \text{nil}) & \leftarrow \\
\text{sorted}(x \cdot y \cdot z) & \leftarrow x \leq y, \text{sorted}(y \cdot z) \\
\text{perm}(\text{nil}, \text{nil}) & \leftarrow \\
\text{perm}(x \cdot y, u \cdot v) & \leftarrow \text{delete}(u, x \cdot y, z), \text{perm}(z, v) \\
\text{delete}(x, x \cdot y, y) & \leftarrow \\
\text{delete}(x, y \cdot z, y \cdot w) & \leftarrow \text{delete}(x, z, w) \\
0 \leq x & \leftarrow \\
s(x) \leq s(y) & \leftarrow x \leq y.
\end{array}$$

No programa, os números inteiros não negativos são representados com termos construídos com 0 e a função sucessor s , onde s é um símbolo de função unário. As potências de s são definidas por indução: $s^0(x) = 0$, $s^{n+1}(x) = s(s^n(x))$. Assim, o número n é representado por $s^n(0)$. Usaremos n para denotar $s^n(0)$ quando for conveniente.

O símbolo “.” é usado na construção de listas (e corresponde à função *cons*) e a lista *nil* representa a lista vazia. Assim, a lista $[4, 22, 11, 14]$ será representada por $4 \cdot (22 \cdot (11 \cdot (14 \cdot \text{nil})))$. Utilizando a convenção usual de associatividade à direita da função “.” podemos escrever simplesmente $4 \cdot 22 \cdot 11 \cdot 14 \cdot \text{nil}$.

O programa *Slowsort* contém as definições dos predicados *sort*, *sorted*, *perm*, *delete*, \leq . Note que o predicado \leq é usado em notação infixa. A semântica informal da *definição* desses predicados é dada por:

- se x e y são listas, e y é uma lista ordenada que é também permutação de x , então y é a versão ordenada de x .
- listas vazias e unitárias são ordenadas.
- listas de dois ou mais elementos são ordenadas se os dois primeiros elementos são mutuamente ordenados e a lista resultante de eliminar o primeiro elemento da lista original é ordenada.
- no caso em que o primeiro elemento da lista y seja x e z seja o resultado de eliminá-lo da lista y , então z é o resultado de eliminar x da lista y e, no caso em que o primeiro elemento da lista y seja diferente de x , o resultado de eliminar x de y será a lista resultante de concatenar o primeiro elemento de y com a lista resultante de eliminar x da lista resultante da lista y sem seu primeiro elemento.
- a lista vazia é uma permutação da lista vazia; a lista x é uma permutação da lista y se ao se retirar o primeiro elemento da lista y , a lista resultante continua sendo uma permutação da lista

obtida a partir de x ao se retirar o mesmo elemento que foi retirado da lista y .

Para executar o programa *slowsort* com entrada $4 \cdot 22 \cdot 11 \cdot 14 \cdot nil$ é preciso apresentar um objetivo da forma $\leftarrow sort(4 \cdot 22 \cdot 11 \cdot 14 \cdot nil, y)$. Isto será interpretado como um requerimento para encontrar um y que seja a versão ordenada de $4 \cdot 22 \cdot 11 \cdot 14 \cdot nil$.

◇

Exemplo 1.2.27 [O problema do fazendeiro] Um fazendeiro quer transportar um lobo, uma galinha e um saco de milho em uma canoa de uma margem à outra de um rio. O problema é que a capacidade da canoa é para dois itens (incluído o próprio fazendeiro) e não é permitido isolar o lobo e galinha nem a galinha e o milho, por motivos óbvios.

Solução usando o sistema *XSB-PROLOG*:

```
demo :- solucone(flgm(oeste,oeste,oeste,oeste), flgm(leste,leste,leste,leste), Sol),
        write_result([[fazend,lobo,galinha,milho]]),write_result(Sol), fail.
solucone( S, G, P ) :- caminho( S, G, [S], P ).
caminho( G, G, H, H ).
caminho( S, G, H, P ) :-
    mover( S, N ),           % mover a um novo estado
    seguro( N ),            % que seja seguro
    nao_repetido( N, H ),   % e nao seja repetido
    caminho( N, G, [N|H], P ). % entao complete o caminho
nao_repetido(N, H) :- repetido(N, H), !, fail.
nao_repetido(_, _).        % temp solution to BA index prob
repetido( X, [X|_] ).
repetido( X, [_|L] ):- repetido( X, L ).
mover( flgm( X, L, G, M ), flgm( Y, L, G, M ) ) :-
    oposto( X, Y ).        % fazendeiro vai sozinho
mover( flgm( X, X, G, M ), flgm( Y, Y, G, M ) ) :-
    oposto( X, Y ).        % fazendeiro vai com o lobo
mover( flgm( X, L, X, M ), flgm( Y, L, Y, M ) ) :-
    oposto( X, Y ).        % fazendeiro vai com a galinha
mover( flgm( X, L, G, X ), flgm( Y, L, G, Y ) ) :-
    oposto( X, Y ).        % fazendeiro vai com o milho
oposto( oeste, leste ).   % lados opostos sao oeste e leste
oposto( leste, oeste ).   % lados opostos sao oeste e leste
seguro( flgm( X, _, X, _ ) ). % fazendeiro e galinha e seguro
seguro( flgm( X, X, _, X ) ). % fazendeiro e lobo e milho e seguro
write_result([]) :- nl.
write_result([X|L]) :- write(X), nl, write_result(L).
```

A seguir o resultado da execução do programa.

```
> xsb -i
XSB Version 1.4.0 (94/5/9)
[sequential, single word, optimal mode]
| ?- [mar_farmer].
[mar_farmer loaded]
yes
| ?- demo.
[ fazend,lobo,galinha,milho]
flgm(leste,leste,leste,leste)
```

```

flgm(oeste,leste,oeste,leste)
flgm(leste,leste,oeste,leste)
flgm(oeste,leste,oeste,oeste)
flgm(leste,leste,leste,oeste)
flgm(oeste,oeste,leste,oeste)
flgm(leste,oeste,leste,oeste)
flgm(oeste,oeste,oeste,oeste)
[ fazend,lobo,galinha,milho]
flgm(leste,leste,leste,leste)
flgm(oeste,leste,oeste,leste)
flgm(leste,leste,oeste,leste)
flgm(oeste,leste,oeste,oeste)
flgm(leste,leste,leste,oeste)
flgm(oeste,oeste,leste,oeste)
flgm(leste,oeste,leste,oeste)
flgm(oeste,oeste,oeste,oeste)
[ fazend,lobo,galinha,milho]
flgm(leste,leste,leste,leste)
flgm(oeste,leste,oeste,leste)
flgm(leste,leste,oeste,leste)
flgm(oeste,oeste,oeste,leste)
flgm(leste,oeste,leste,leste)
flgm(oeste,oeste,leste,oeste)
flgm(leste,oeste,leste,oeste)
flgm(oeste,oeste,oeste,oeste)
[ fazend,lobo,galinha,milho]
flgm(leste,leste,leste,leste)
flgm(oeste,leste,oeste,leste)
flgm(leste,leste,oeste,leste)
flgm(oeste,oeste,oeste,leste)
flgm(leste,oeste,leste,leste)
flgm(oeste,oeste,leste,oeste)
flgm(leste,oeste,leste,oeste)
flgm(oeste,oeste,oeste,oeste)
yes
| ?- halt.
End XSB (cputime 0.51s, elapsetime 12.98s)
>

```

◇

1.3 Interpretações e Modelos

A semântica declarativa dos programas lógicos é dada pela semântica usual das fórmulas da lógica de primeira-ordem. Discutir-se-ão os modelos e interpretações dos programas lógicos e em particular as interpretações de Herbrand que tratam de corresponder às interpretações “desejadas” dos programas lógicos.

As interpretações são simplesmente um domínio de discurso onde são “interpretados” símbolos de constante, função e predicado. Assim, as constantes são “interpretadas” como elementos do domínio, símbolos de função, como funções no domínio e símbolos de predicado como predicados no domínio (da correspondente aridade).

Uma interpretação expressa um significado por cada símbolo dentro das fórmulas. São de interesse, em particular, interpretações que validam as fórmulas, i.e. **modelos** das fórmulas. De maneira geral, existe uma **interpretação de intenção** C (a que o especificador do programa imagina estar especificando) que fornece o principal significado às fórmulas e deve ser um modelo das fórmulas.

A lógica de primeira-ordem provê métodos para deduzir os teoremas de uma teoria; i.e., as fórmulas que são válidas em qualquer modelo da teoria. Tais teoremas podem ser caracterizados pelo teorema de completude de Gödel, como as fórmulas que são **consequências lógicas** dos axiomas da teoria. Em outras palavras, como as fórmulas que valem em cada interpretação que é um modelo de cada um dos axiomas da teoria.

Os axiomas de um programa lógico, são as declarações do programa.

Suponha que se deseja provar que a fórmula:

$$\exists_{y_1} \dots \exists_{y_r} (B_1 \wedge \dots \wedge B_n)$$

é uma consequência lógica de um programa P . A negação da fórmula a ser provada é adicionada aos axiomas do programa e trata-se de derivar uma contradição, ou seja a prova construída é **refutacional**. Se é possível obter uma contradição da negação da fórmula, então a validade desta é atingida. A negação da fórmula a ser provada é o objetivo:

$$\leftarrow B_1, \dots, B_n$$

Se derivando novos objetivos a partir do anterior obtém-se a cláusula vazia, então tem-se uma contradição, o que implica que $\exists_{y_1, \dots, y_r} (B_1 \wedge \dots \wedge B_n)$ é uma consequência lógica de P .

Do ponto de vista da demonstração automática de teoremas, o interessante é demonstrar a validade de $\exists_{y_1, \dots, y_r} (B_1 \wedge \dots \wedge B_n)$, mas do ponto de vista do programador, interessantes são as ligações das variáveis $y_1 \dots y_r$ que foram necessárias para atingir a prova. Em outras palavras, os passos construtivos da prova refutacional. Tais ligações, podem-se considerar como a **saída** do programa.

Exemplo 1.3.1 Considerando novamente o programa *slowsort* do exemplo 1.2.26, o objetivo

$$\leftarrow \text{sort}(4 \cdot 22 \cdot 11 \cdot 14 \cdot \text{nil}, y)$$

é simplesmente um requerimento para demonstrar que $\exists_y \text{sort}(4 \cdot 22 \cdot 11 \cdot 14 \cdot \text{nil}, y)$ é uma consequência lógica do programa. De fato, o principal resultado não é a validade da fórmula, mas a ligação da variável y que deve induzir a uma ordenação da lista $4 \cdot 22 \cdot 11 \cdot 14 \cdot \text{nil}$. \diamond

Definição 1.3.2 Uma **pré-interpretação**, \mathcal{J} , de uma linguagem de primeira-ordem \mathcal{L} consiste do seguinte:

- a) Um conjunto **não-vazio** D denominado o domínio da pré-interpretação.
- b) Para cada constante em \mathcal{L} , a designação de um elemento em D .
- c) Para cada símbolo de função n -ária em \mathcal{L} , a designação de uma função de D^n para D .

Definição 1.3.3 Uma **interpretação** \mathcal{I} de uma linguagem de primeira-ordem \mathcal{L} consiste de uma pré-interpretação \mathcal{J} , com domínio D , conjuntamente com o seguinte:

Para cada símbolo de predicado n -ário em \mathcal{L} , a designação de uma função de D^n no conjunto $\{true, false\}$ (ou equivalentemente uma relação sobre D^n).

Diz-se que \mathcal{I} é baseada em \mathcal{J} .

Definição 1.3.4 Seja \mathcal{J} uma pré-interpretação de uma linguagem de primeira-ordem \mathcal{L} . Uma **designação de variáveis** \mathcal{V} (com respeito a \mathcal{J}) é uma designação para cada variável em \mathcal{L} de um elemento no domínio de \mathcal{J} .

Notação: Uma designação de variáveis é especificada como um conjunto de ligações entre variáveis da linguagem e elementos do domínio: $\mathcal{V} = \{x_1/d_1, \dots, x_n/d_n, \dots\}$. O resultado de aplicar as designações de uma ligação de variáveis numa fórmula (ou termo) W é denotado por $W\mathcal{V}$. •

Definição 1.3.5 Seja \mathcal{J} uma pré-interpretação de uma linguagem de primeira-ordem \mathcal{L} e seja \mathcal{V} uma designação de variáveis. A **designação de termos** (segundo \mathcal{J} e \mathcal{V}) dos termos em \mathcal{L} é definida como segue:

- a) Cada variável assume sua designação segundo \mathcal{V} .
- b) Cada constante assume sua designação segundo \mathcal{J} .
- c) Se d_1, \dots, d_n são as correspondentes designações dos termos t_1, \dots, t_n e f' é a função de D^n em D (domínio de \mathcal{J}) associada ao símbolo de função n -ário f em \mathcal{L} , então $f'(d_1, \dots, d_n)$ é a designação correspondente ao termo $f(t_1, \dots, t_n)$ (segundo \mathcal{J} e \mathcal{V}).

Definição 1.3.6 Sejam \mathcal{J} uma pré-interpretação, \mathcal{V} uma designação das variáveis com respeito a \mathcal{J} e A um átomo. Suponha A é $p(t_1, \dots, t_n)$ e d_1, \dots, d_n são as designações dos termos t_1, \dots, t_n , respectivamente, com respeito a \mathcal{J} e \mathcal{V} .

A \mathcal{J} -instância de A com respeito a \mathcal{V} é $p(d_1, \dots, d_n) = A_{\mathcal{J}, \mathcal{V}}$. Seja

$$[A]_{\mathcal{J}} = \{A_{\mathcal{J}, \mathcal{V}} \mid \mathcal{V} \text{ designação de variáveis com respeito a } \mathcal{J}\}.$$

Cada elemento de $[A]_{\mathcal{J}}$ é denominado uma \mathcal{J} -instância de A . Cada $p(d_1, \dots, d_n)$ é também chamado uma \mathcal{J} -instância.

Definição 1.3.7 Seja \mathcal{I} uma interpretação com domínio D de uma linguagem de primeira-ordem \mathcal{L} e seja \mathcal{V} uma designação das variáveis. Então uma fórmula em \mathcal{L} tem um **valor de verdade** *true* ou *false* (com respeito a \mathcal{I} e \mathcal{V}) segundo o seguinte:

- a) Se a fórmula é um átomo $p(t_1, \dots, t_n)$, então o valor de verdade é obtido calculando o valor de $p'(d_1, \dots, d_n)$, onde p' é a função designada à p por \mathcal{I} e d_1, \dots, d_n são as designações dos termos t_1, \dots, t_n com respeito a \mathcal{I} e \mathcal{V} .
- b) Se a fórmula tem a forma $\neg F, F \wedge G, F \vee G, F \rightarrow G, F \leftrightarrow G$, então o valor de verdade é dado pela seguinte tabela:

F	G	$\neg F$	$F \wedge G$	$F \vee G$	$F \rightarrow G$	$F \leftrightarrow G$
<i>true</i>	<i>true</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>
<i>true</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>
<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>
<i>false</i>	<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>true</i>

- c) Se a fórmula tem a forma $\exists_x F$, então seu valor de verdade é *true* se existe $d \in D$ tal que F tem valor de verdade *true* com respeito a \mathcal{I} e $\mathcal{V}(x/d)$, onde $\mathcal{V}(x/d)$ é \mathcal{V} exceto que x tem designação d . Em caso contrário o valor de verdade da fórmula é *false*.
- d) Se a fórmula tem a forma $\forall_x F$, então seu valor de verdade é *true* se para todo $d \in D$, F tem valor de verdade *true* com respeito a \mathcal{I} e $\mathcal{V}(x/d)$. Em caso contrário seu valor de verdade é *false*.

Observação: note que o valor de verdade de uma sentença não depende da designação dada às variáveis. Consequentemente pode-se falar do *valor de verdade de uma sentença com respeito a uma interpretação* \mathcal{I} . Se o valor de verdade de uma sentença com respeito a \mathcal{I} é *true* (respectivamente *false*), diz-se que a fórmula é *true* (respectivamente *false*) com respeito a \mathcal{I} . Por simplicidade usar-se-ão também as expressões: “a fórmula F é verdadeira (resp. falsa) em \mathcal{I} ” ou “a fórmula é válida (resp. não vale) em \mathcal{I} ” sempre que o contexto seja claro, pois a palavra *validade* tem um significado reservado muito especial em lógica matemática como será apresentado nas seguintes definições. •

Definição 1.3.8 *Sejam \mathcal{I} uma interpretação de uma linguagem de primeira-ordem \mathcal{L} e W uma fórmula em \mathcal{L} .*

- W é **consistente** ou **satisfatível** em \mathcal{I} se $\exists W$ é true com respeito a \mathcal{I} .
- W é **válida** em \mathcal{I} se $\forall W$ é true com respeito a \mathcal{I} .
- W é **inconsistente** ou **insatisfatível** em \mathcal{I} se $\exists W$ é false com respeito a \mathcal{I} .
- W é **inválida** em \mathcal{I} se $\forall W$ é false com respeito a \mathcal{I} .

Definição 1.3.9 *Seja \mathcal{I} uma interpretação de uma linguagem de primeira-ordem \mathcal{L} e seja F uma sentença de \mathcal{L} . Então \mathcal{I} é um **modelo** de F se F é true com respeito a \mathcal{I} .*

Exemplo 1.3.10 Considere a fórmula $\forall_x \exists_y p(x, y)$ e a seguinte interpretação \mathcal{I} :

- o domínio de \mathcal{I} é o conjunto dos inteiros não negativos;
- a designação de p é a relação $<$.

Então \mathcal{I} é um modelo da fórmula. Em \mathcal{I} a fórmula expressa que para todo inteiro não negativo existe outro inteiro estritamente maior.

Note que outras interpretações são possíveis; por exemplo, \mathcal{I}' cujo domínio é o conjunto dos inteiros negativos e a mesma designação para o símbolo de predicado p . Então a fórmula é *falsa* com respeito a \mathcal{I}' (ou equivalentemente, \mathcal{I}' não é modelo da fórmula). \diamond

Os *axiomas* de uma teoria de primeira-ordem são um conjunto dado de sentenças da linguagem. As teorias que se tratam na programação lógica têm cláusulas de programas como axiomas.

Definição 1.3.11 *Seja T uma teoria de primeira-ordem e seja \mathcal{L} a linguagem de T . Um **modelo** de T é uma interpretação de \mathcal{L} que é um modelo de cada axioma de T .*

No caso em que T tenha um modelo diz-se que T é consistente.

Definição 1.3.12 *Seja S um conjunto de sentenças de uma linguagem de primeira-ordem \mathcal{L} e seja \mathcal{I} uma interpretação de \mathcal{L} . \mathcal{I} é **modelo** de S se \mathcal{I} é modelo de cada fórmula de S .*

Observação: Note que se $S = \{F_1, \dots, F_n\}$ é um conjunto finito de sentenças então \mathcal{I} é modelo de S sse o é de $F_1 \wedge \dots \wedge F_n$. \bullet

Definição 1.3.13 *Seja S um conjunto de sentenças de uma linguagem de primeira-ordem \mathcal{L} .*

- S é **satisfatível** se \mathcal{L} tem uma interpretação que é modelo de S .
- S é **válido** se cada interpretação de \mathcal{L} é modelo de S .
- S é **insatisfatível** ou **inconsistente** se nenhuma interpretação de \mathcal{L} é modelo de S .
- S é **inválido** se \mathcal{L} tem uma interpretação que não é modelo de S .

Exemplo 1.3.14 Conjuntos satisfatíveis.

1. Seja $S_1 = \{\forall x \exists y p(x, y)\}$. S_1 é tanto satisfatível como inválido, já que p pode ser interpretada como a relação menor no domínio dos números naturais ou como a relação “pai” no domínio das pessoas.
2. Os monóides são os modelos do seguinte conjunto de axiomas

$$S_2 = \left\{ \begin{array}{l} \forall x eq(e \cdot x, x), \\ \forall x eq(x \cdot e, x), \\ \forall x \forall y \forall z eq((x \cdot y) \cdot z, x \cdot (y \cdot z)), \\ \forall x \forall y (eq(x, y) \leftrightarrow eq(y, x)), \forall x \forall y \forall z (eq(x, y) \rightarrow (eq(x \cdot z, y \cdot z) \wedge eq(z \cdot x, z \cdot y))) \end{array} \right\}$$

Basta interpretar e como o elemento neutro, eq como a relação de igualdade e \cdot como o operador multiplicativo do monóide. As duas últimas fórmulas de S_2 expressam a simetria e reflexividade funcional (com respeito ao símbolo de função \cdot) do símbolo de predicado eq . O mais usual, quando é necessário trabalhar com o predicado de igualdade, é usar linguagens de primeira ordem com igualdade, onde o predicado de igualdade é pré-interpretado evitando dessa maneira a apresentação explícita dos axiomas de igualdade.

◇

Definição 1.3.15 Seja S um conjunto de sentenças e F uma sentença de uma linguagem de primeira ordem \mathcal{L} . Diz-se que F é uma **consequência lógica** de S se para cada interpretação \mathcal{I} de \mathcal{L} que é modelo de S , \mathcal{I} é modelo de F .

Observação: note que se F é consequência lógica de $\{F_1, \dots, F_n\}$, então $F_1 \wedge \dots \wedge F_n \rightarrow F$ é válida. A recíproca também é certa: se $F_1 \wedge \dots \wedge F_n \rightarrow F$ é válida, então toda interpretação que seja modelo de $\{F_1, \dots, F_n\}$ é também modelo de F . •

Exemplo 1.3.16 Continuação exemplo 1.3.14.2. Seja $S_3 = S_2 \cup \{\forall x eq(x \cdot x, e)\}$. Os monóides nilpotentes são os modelos de S_3 . Observe que $\forall x \forall y eq(x \cdot y, y \cdot x)$ é uma consequência lógica de S_3 , já que qualquer monóide nilpotente é abeliano. ◇

Proposição 1.3.17 *Seja S um conjunto de sentenças e F uma sentença de uma linguagem de primeira-ordem \mathcal{L} . Então F é consequência lógica de S sse $S \cup \{\neg F\}$ é insatisfatível.*

Demonstração. Suponha que F é consequência lógica de S e seja \mathcal{I} uma interpretação de \mathcal{L} que é modelo de S . Então \mathcal{I} é também modelo de F , portanto \mathcal{I} não é modelo de $S \cup \{\neg F\}$. Logo $S \cup \{\neg F\}$ é insatisfatível.

No outro sentido, se $S \cup \{\neg F\}$ é insatisfatível, seja \mathcal{I} uma interpretação de \mathcal{L} e suponha que é modelo de S . Como $S \cup \{\neg F\}$ é insatisfatível, \mathcal{I} não é modelo de $\neg F$. Consequentemente \mathcal{I} é modelo de F . Logo F é consequência lógica de S . \square

Exemplo 1.3.18 Sejam $S = \{p(a), \forall x(p(x) \rightarrow q(x))\}$ e $F = q(a)$. F é consequência lógica de S ; com efeito, seja \mathcal{I} um modelo de S , então $p(a)$ é *true* com respeito a \mathcal{I} e como também $\forall x(p(x) \rightarrow q(x))$ é *true* com respeito a \mathcal{I} , em particular $p(a) \rightarrow q(a)$ é *true* com respeito a \mathcal{I} . Portanto $q(a)$ é *true* com respeito a \mathcal{I} ou, em outras palavras, \mathcal{I} é também modelo de $q(a)$. \diamond

Usando as definições dadas até o momento, podemos formalizar o processo de execução dos programas lógicos da seguinte maneira: dado um programa P (conjunto de cláusulas de programa) e um objetivo G , trata-se de demonstrar que o conjunto de cláusulas $P \cup \{G\}$ é insatisfatível. Com efeito, se G é o objetivo $\leftarrow B_1, \dots, B_n$ com variáveis y_1, \dots, y_r , então pela proposição anterior a insatisfatibilidade de $P \cup \{G\}$ é equivalente a dizer que $\neg G$ é consequência lógica de P . Note que $\neg(\forall y_1 \dots \forall y_r(\leftarrow B_1, \dots, B_n))$ é *equivalente* a $\exists y_1 \dots \exists y_r(B_1 \wedge \dots \wedge B_n)$.

O problema de demonstrar a insatisfatibilidade de $P \cup \{G\}$ é o mesmo de demonstrar que toda interpretação da linguagem \mathcal{L} não é modelo de $P \cup \{G\}$. As interpretações de Herbrand serão importantes pois representam uma classe de interpretações suficientes para tratar o anterior problema de inconsistência. Não será preciso, então, testar a validade ou invalidade de $P \cup \{G\}$ em todas as interpretações.

Definição 1.3.19 *Um termo básico (termo ground) é um termo sem variáveis.*

Definição 1.3.20 *Seja \mathcal{L} uma linguagem de primeira-ordem. O universo de Herbrand $U_{\mathcal{L}}$ para \mathcal{L} é o conjunto de todos os termos básicos que podem ser construídos usando os símbolos de constante e funções de \mathcal{L} . No caso em que \mathcal{L} não tenha constantes, inclui-se uma nova constante para formar os termos básicos.*

Exemplo 1.3.21 Considere a linguagem \mathcal{L} que tem símbolos de função unária f e binária g . O universo de Herbrand da linguagem \mathcal{L} será:

$$\{a, f(a), g(a, a), f(f(a)), f(g(a, a)), g(a, f(a)), g(f(a), a), g(f(a), f(a)), g(a, g(a, a)), \dots\}$$

onde a é um novo símbolo de constante.

No caso em que a linguagem \mathcal{L} contém símbolos de constante, por exemplo b, c , o universo de Herbrand será construído usando tais símbolos de constante:

$$\{b, c, f(b), f(c), g(b, b), g(c, c), g(b, c), g(c, b), f(f(b)), \dots\}$$

◇

Definição 1.3.22 *Seja \mathcal{L} uma linguagem de primeira-ordem. A base de Herbrand $B_{\mathcal{L}}$ de \mathcal{L} é o conjunto de todas as fórmulas atômicas formadas com símbolos de predicados de \mathcal{L} e termos do universo de Herbrand de \mathcal{L} .*

Usualmente falar-se-á de átomos básicos (ou *ground*).

Exemplo 1.3.23 Continuando o exemplo anterior, se a linguagem \mathcal{L} tem símbolos de predicado unário p e binário r , então, $p(a), p(f(g(a, f(a))))$, $r(a, f(a))$, etc. são átomos básicos na base de Herbrand de \mathcal{L} .

◇

Definição 1.3.24 *Seja \mathcal{L} uma linguagem de primeira-ordem. A pré-interpretação de Herbrand de \mathcal{L} é a pré-interpretação definida como segue:*

- a) *O domínio da pré-interpretação é o universo de Herbrand $U_{\mathcal{L}}$.*
- b) *A designação para as constantes em \mathcal{L} são as mesmas constantes.*
- c) *Se f é um símbolo de função n -ária em \mathcal{L} , então o mapeio designado para f , de $U_{\mathcal{L}}^n$ em $U_{\mathcal{L}}$ é definido por $(t_1, \dots, t_n) \mapsto f(t_1, \dots, t_n)$.*

Uma **interpretação de Herbrand** para \mathcal{L} é qualquer interpretação baseada na pré-interpretação de Herbrand de \mathcal{L} .

Definição 1.3.25 *Seja \mathcal{L} uma linguagem de primeira-ordem e S um conjunto de sentenças de \mathcal{L} . Um modelo de Herbrand para S é uma interpretação de Herbrand de \mathcal{L} que seja modelo de S .*

Nota: Usualmente será usada a expressão *interpretação de S* e não interpretação da linguagem \mathcal{L} de S . Assim, poder-se-á falar do universo de Herbrand de S , U_S , da base de Herbrand de S , B_S , e da interpretação de Herbrand de S . Normalmente isto é possível, já que a linguagem \mathcal{L} pode ser deduzida dos símbolos usados pelas fórmulas em S , que geralmente são as fórmulas clausais do programa. Será, então, usual falar do universo de Herbrand do programa P , U_P , da base de Herbrand do programa P , B_P , e de fato, da interpretação de Herbrand do programa P .

•

Exemplo 1.3.26 Continuação do exemplo 1.2.26. A linguagem usada pelo programa *slowsort* inclui as constantes 0 e nil , símbolos de função s e \cdot e símbolos de predicado $sort$, $perm$, $sorted$, $delete$ e \leq . A interpretação de intenção é uma interpretação de Herbrand. Um átomo $sort(l, m)$ é da interpretação de intenção sse tanto l como m são nil ou uma lista de termos da forma $s^k(0)$ e m é a versão ordenada de l . Aos outros símbolos de predicados são designadas as relações óbvias. A interpretação de intenção é um modelo do programa e para a teoria associada.

Note que o universo de Herbrand é conformado pelos conjuntos:

$$\{0, s(0), s^2(0), \dots\} \cup \{nil, 0 \cdot nil, 1 \cdot nil, 0 \cdot 0 \cdot nil, 0 \cdot 1 \cdot nil, 0 \cdot 2 \cdot nil, 1 \cdot 0 \cdot nil, 1 \cdot 1 \cdot nil, 1 \cdot 2 \cdot nil, \dots\},$$

onde $1, 2, \dots$ são abreviações de $s(0), s(s(0)), \dots$, respectivamente. \diamond

A seguinte proposição confere o fato de que para demonstrar a insatisfatibilidade de um conjunto de cláusulas é suficiente considerar as interpretações de Herbrand.

Proposição 1.3.27 *Seja S um conjunto de cláusulas e suponha que S tem um modelo. Então S tem um modelo de Herbrand.*

Demonstração. Seja \mathcal{I} uma interpretação de S . Define-se uma interpretação de Herbrand \mathcal{I}' , associada com \mathcal{I} da seguinte maneira:

- Para cada fórmula atômica $p(t_1, \dots, t_n)$ da base de Herbrand B_S , $p(t_1, \dots, t_n)$ é *true* em \mathcal{I}' sse $p(t_1, \dots, t_n)$ é *true* com respeito a \mathcal{I} .

Agora vejamos que se \mathcal{I} é um modelo, então também \mathcal{I}' é um modelo de S :

Seja $\forall x_1, \dots, \forall x_l (A_1 \vee \dots \vee A_k \vee \neg B_1 \vee \dots \vee \neg B_m)$ uma cláusula em S . Então, se \mathcal{I} é um modelo de S e $\mathcal{V} = \{x_1/t_1, \dots, x_l/t_l\}$ uma designação das variáveis da cláusula em termos básicos,

$$(A_1 \vee \dots \vee A_k \vee \neg B_1 \vee \dots \vee \neg B_m)\mathcal{V}$$

é verdadeiro em \mathcal{I} , o que implica que algum literal positivo $A_j\mathcal{V}$ ou algum literal negativo $\neg B_i\mathcal{V}$ é verdadeiro em \mathcal{I} . Caso $A_j\mathcal{V}$ seja verdadeiro, pela definição de \mathcal{I}' , $A_j\mathcal{V}$ é também verdadeiro em \mathcal{I}' . Caso $\neg B_i\mathcal{V}$ seja verdadeiro em \mathcal{I} , $B_i\mathcal{V}$ será falso em \mathcal{I}' . Logo $\neg B_i\mathcal{V}$ será verdadeiro em \mathcal{I}' . Como \mathcal{V} foi selecionada aleatoriamente, pode-se concluir que a cláusula

$$\forall x_1, \dots, \forall x_l (A_1 \vee \dots \vee A_k \vee \neg B_1 \vee \dots \vee \neg B_m)$$

é verdadeira em \mathcal{I}' . Assim, conclui-se que \mathcal{I}' é um modelo de S . \square

Nota: É importante notar que a forma restrita das cláusulas permite a demonstração da proposição.

Para fórmulas gerais, isto não é possível. Por exemplo, se temos que $\exists_x F$ é verdadeira em \mathcal{I} , para demonstrar indutivamente que a mesma vale em \mathcal{I}' , temos o problema de que uma “testemunha” para o quantificador existencial não será necessariamente um termo básico. Nesse caso será preciso supor que trabalhamos com teorias de Henkin, onde para qualquer sentença existencial $\phi = \exists_x F$ existe uma constante c_ϕ tal que a fórmula $\exists_x F \rightarrow F(x/c_\phi)$ é verdadeira. Isto é usado na demonstração da completude da lógica de primeira-ordem. Veja por exemplo [CK90]. •

Proposição 1.3.28 *Seja S um conjunto de cláusulas. Então S é inconsistente sse S não tem um modelo de Herbrand.*

Demonstração. Consequência direta da proposição anterior. □

Exemplo 1.3.29 Para ilustrar o fato de que as proposições anteriores são certas só para cláusulas (mencionado na nota anterior) considere o conjunto de fórmulas

$$S = \{p(a), \exists_x \neg p(x)\}$$

Um modelo de S tem um domínio com pelo menos dois elementos. Por exemplo $D = \{0, 1\}$, onde a designação de a é 0 (p.ex.) e a de p é a relação unitária que inclui 0 (i.e., $p(0)$ é *true*, e $p(1)$ é *false*).

Note que não temos um modelo de Herbrand já que $U_S = \{a\}$. Logo ambas as fórmulas $p(a)$ e $\exists_x \neg p(x)$ não podem ser válidas para nenhuma interpretação de Herbrand de S . ◇

Quando seja preciso tratar fórmulas não clausais será necessário considerar interpretações arbitrárias e não somente de Herbrand.

Definição 1.3.30 *Uma fórmula em forma normal prenexa conjuntiva tem a seguinte estrutura:*

$$Q_{x_1}^1, \dots, Q_{x_k}^k ((L_{11} \vee \dots \vee L_{1m_1}) \wedge \dots \wedge (L_{n1} \vee \dots \vee L_{nm_n})),$$

onde cada Q^i é um quantificador existencial ou universal e os L_{ij} 's são literais.

Definição 1.3.31 *Duas fórmulas V e W são logicamente equivalentes se $\forall(V \leftrightarrow W)$ é válida.*

Proposição 1.3.32 *Para cada fórmula W , existe uma fórmula logicamente equivalente em forma normal prenexa conjuntiva.*

Demonstração. (*Rascunho*) A demonstração é baseada em uma série de transformações aplicáveis às fórmulas que não trocam a semântica da fórmula original.

- Fórmulas da forma $F \rightarrow G$ podem ser transformadas em $\neg F \vee G$.

- Fórmulas da forma $F \leftrightarrow G$ em $(\neg F \vee G) \wedge (F \vee \neg G)$.
- A negação de fórmulas compostas pode ser tratada pelas seguintes transformações:

$$\begin{array}{lll} \neg \forall_x F & \text{em} & \exists_x \neg F \\ \neg \exists_x F & \text{em} & \forall_x \neg F \\ \neg(F \vee G) & \text{em} & \neg F \wedge \neg G \\ \neg(F \wedge G) & \text{em} & \neg F \vee \neg G \\ \neg \neg F & \text{em} & F \end{array}$$

até que cada negação ocorra precedendo um átomo.

- Os quantificadores dentro de fórmulas compostas podem ser transferidos ao início das fórmulas pelas seguintes transformações:

$$\begin{array}{lll} Q_x F \vee G & \text{em} & Q_x(F \vee G) \\ G \vee Q_x F & \text{em} & Q_x(G \vee F) \\ Q_x F \wedge G & \text{em} & Q_x(F \wedge G) \\ G \wedge Q_x F & \text{em} & Q_x(G \wedge F) \end{array}$$

onde G não depende da variável x .

$$\begin{array}{lll} \exists_x F \vee \exists_x G & \text{em} & \exists_x(F \vee G) \\ \forall_x F \wedge \forall_x G & \text{em} & \forall_x(F \wedge G) \end{array}$$

No caso em que a variável quantificada ocorra fora do campo de abrangência do quantificador será necessário trocar a variável por outra que não gere conflitos.

- Finalmente conectivos \vee 's e \wedge 's podem ser distribuídos obtendo formas conjuntivas como segue:

$$\begin{array}{lll} (F \wedge G) \vee H & \text{em} & (F \vee H) \wedge (G \vee H) \\ H \vee (F \wedge G) & \text{em} & (H \vee F) \wedge (H \vee G). \end{array}$$

As transformações mantêm a semântica da fórmula original, já que as fórmulas em cada passo da transformação são logicamente equivalentes. □

Exemplo 1.3.33 Considere a transformação em forma normal prenexa conjuntiva da fórmula:

$$\begin{array}{l} \neg p(x) \rightarrow (\neg \forall_x (r(z, x) \vee \neg p(x))) \\ \leftrightarrow \neg \neg p(x) \vee (\neg \forall_x (r(z, x) \vee \neg p(x))) \\ \leftrightarrow p(x) \vee (\exists_x \neg (r(z, x) \vee \neg p(x))) \\ \leftrightarrow p(x) \vee (\exists_x (\neg r(z, x) \wedge \neg \neg p(x))) \\ \leftrightarrow \exists_w (p(x) \vee (\neg r(z, w) \wedge p(w))) \\ \leftrightarrow \exists_w ((p(x) \vee \neg r(z, w)) \wedge (p(x) \vee p(w))) \end{array}$$

◇

Em aplicações reais (como é o caso de *slowsort*, onde temos dois tipos ou *sortes* de objetos: inteiros e listas) o domínio de interpretação de um programa lógico deve incluir uma família de domínios de diferentes tipos. Para estes casos as definições de símbolos de função e de predicados devem indicar o “tipo de aridade” e não uma aridade numérica como no caso de teorias com um único tipo (i.e., as estudadas até o momento). Por exemplo o predicado *delete* do programa *slowsort* tem como argumentos um objeto de tipo numérico e dois de tipo listas de objetos.

Teorias com diferentes tipos são denominadas **teorias tipadas** ou **teorias poli-sortidas** de primeira-ordem. Tais teorias incluem um conjunto finito de **tipos** denotados por letras gregas. Cada símbolo de variável, constante, função, predicado e quantificador é tipado. O tipo de uma variável ou de uma constante é um elemento τ do conjunto de tipos. O tipo de um símbolo de predicado é uma n -tupla de elementos do conjunto de tipos, denotado por $\tau_1 \times \dots \times \tau_n$ e os símbolos de função têm tipos da forma $\tau_1 \times \dots \times \tau_n \rightarrow \tau$, onde τ é o tipo da função ou tipo do codomínio da função e $\tau_1 \times \dots \times \tau_n$ o do seu domínio. Para cada tipo τ no conjunto de tipos existe um quantificador existencial e um universal, respectivamente: \forall^τ , \exists^τ . Geralmente os subíndices de tipo e definições de tipo de variáveis, constantes, funções e predicados são omitidos, sempre que, usualmente, se deduzem das fórmulas. Apresentar-se-ão algumas definições para teorias tipadas com o objetivo de ilustrar as considerações necessárias.

Definição 1.3.34 *Um termo de tipo τ é definido indutivamente como segue:*

- a) *Uma variável de tipo τ é um termo de tipo τ .*
- b) *Uma constante de tipo τ é um termo de tipo τ .*
- c) *Se f é uma função de tipo $\tau_1 \times \dots \times \tau_n \rightarrow \tau$ e t_i é um termo de tipo τ_i , $i = 1, \dots, n$, então $f(t_1, \dots, t_n)$ é um termo de tipo τ .*

Definição 1.3.35 *Uma fórmula tipada (bem formada) é definida indutivamente como segue:*

- a) *Se p é um predicado de tipo $\tau_1 \times \dots \times \tau_n$ e t_i é um termo de tipo τ_i , $i = 1, \dots, n$, então $p(t_1, \dots, t_n)$ é uma fórmula tipada (atômica).*
- b) *Se F e G são fórmulas tipadas, então $\neg F$, $F \wedge G$, $F \vee G$, $F \rightarrow G$, $F \leftrightarrow G$ são fórmulas tipadas.*
- c) *Se F é uma fórmula tipada e x é uma variável de tipo τ , então $\forall_x^\tau F$ e $\exists_x^\tau F$ são fórmulas tipadas.*

Definição 1.3.36 *A linguagem de primeira-ordem tipada consiste do conjunto de todas as fórmulas tipadas construídas com símbolos do alfabeto em questão.*

Definição 1.3.37 Uma **pré-interpretação**, \mathcal{J} , de uma linguagem de primeira-ordem tipada, \mathcal{L} , consiste de:

- a) Para cada tipo τ em \mathcal{L} , um conjunto não vazio D_τ chamado o **domínio de tipo** τ da pré-interpretação.
- b) Para cada constante de tipo τ em \mathcal{L} , uma designação de um elemento em D_τ .
- c) Para cada função em \mathcal{L} de tipo $\tau_1 \times \dots \times \tau_n \rightarrow \tau$, uma designação de uma função de $D_{\tau_1} \times \dots \times D_{\tau_n}$ para D_τ .

Definição 1.3.38 Uma **interpretação**, \mathcal{I} , de uma linguagem de primeira-ordem tipada \mathcal{L} consiste de uma pré-interpretação \mathcal{J} com domínio de tipo D_τ para cada τ no conjunto de tipos, com o seguinte:

- para cada símbolo de predicado de tipo $\tau_1 \times \dots \times \tau_n$ em \mathcal{L} , a designação de uma função de $D_{\tau_1} \times \dots \times D_{\tau_n}$ no conjunto $\{\text{true}, \text{false}\}$ ou equivalentemente uma relação em $D_{\tau_1} \times \dots \times D_{\tau_n}$. Diz-se que \mathcal{I} é baseada na pré-interpretação \mathcal{J} .

Outras definições necessárias, como por exemplo designação de variáveis, de termos, valores de verdade, consequência lógica, modelo, etc., podem ser construídas de maneira análoga para teorias de primeira-ordem tipadas.

Observação: O uso de fórmulas tipadas é elegante e simplifica o trabalho de especificação de programas (correspondentemente, de axiomatização de teorias), mas não é essencial, já que qualquer teoria tipada pode ser transformada em uma teoria livre de tipos [Gal87]. Assim, todos os resultados obtidos para teorias não tipadas são válidos automaticamente para as teorias tipadas. •

1.4 Skolemização: relação entre a lógica de primeira-ordem e a clausal

O processo de skolemização é simplesmente a substituição dos quantificadores existenciais nas sentenças por novas funções “testemunhas” da existência dos objetos das interpretações do modelo original. O nome Skolemização é devido ao inventor do processo, Skolem.

A idéia básica do processo de Skolemização é conceber os quantificadores existenciais como abreviações da linguagem.

Exemplo 1.4.1 Considere a fórmula

$$\forall x \exists y (x + x = y)$$

O quantificador existencial pode-se considerar um mecanismo para abreviar os símbolos utilizados na linguagem. Poderíamos também pensar em expressar a fórmula anterior como

$$\forall_x f(x) = x + x$$

Neste caso, trata-se de definir a função $f(x)$, a ser interpretada como o dobro de x (sempre que o símbolo “+” seja interpretado como a adição!). \diamond

Nesta seção a fórmula normal prenexa conjuntiva de uma sentença F será denotada da seguinte maneira:

$$F_{fnp} = Q_{x_1}^1, \dots, Q_{x_k}^k (C_1 \wedge \dots \wedge C_n),$$

onde os C_i 's são disjunções de literais da forma $(L_{i1} \vee \dots \vee L_{im_i})$. Para maior simplicidade a *matriz*, $(C_1 \wedge \dots \wedge C_n)$, da forma normal prenexa conjuntiva será denotada por $M[x_1, \dots, x_k]$.

Definição 1.4.2 A *skolemização* de uma sentença em forma prenexa da forma

$$Q_{x_1}^1, \dots, Q_{x_k}^k M[x_1, \dots, x_k]$$

é construída sequencialmente transformando a fórmula até que todos os quantificadores existenciais sejam eliminados da seguinte maneira:

- Seja $Q_{x_j}^j$ o primeiro quantificador existencial da fórmula (da esquerda para a direita) e seja f_{x_j} um novo símbolo de função $(j - 1)$ -ário. A fórmula se transforma na seguinte:

$$Q_{x_1}^1, \dots, Q_{x_{j-1}}^{j-1}, Q_{x_{j+1}}^{j+1}, \dots, Q_{x_k}^k M[x_1, \dots, x_{j-1}, x_j/f_{x_j}(x_1, \dots, x_{j-1}), x_{j+1}, \dots, x_k]$$

Observação: note que a Skolemização de uma fórmula em forma normal prenexa conjuntiva é um conjunto de cláusulas \bullet

Notação: Seja F uma sentença. A skolemização de F_{fnp} é denotada por F_{Sk} \bullet

Teorema 1.4.3 *Seja F uma sentença da lógica de primeira-ordem. F é inconsistente sse F_{Sk} é inconsistente.*

Demonstração. Pela proposição 1.3.32 basta demonstrar que F_{fnp} é inconsistente sse F_{Sk} o é. Considerar-se-á um único passo da transformação e demonstrar-se-á que após um passo a fórmula resultante é inconsistente sse F_{fnp} o é.

Seja $F_{fnp} = Q_{x_1}^1, \dots, Q_{x_k}^k \ M[x_1, \dots, x_k]$ a forma normal prenexa conjuntiva de F e seja

$$F_{Sk_1} = Q_{x_1}^1, \dots, Q_{x_{j-1}}^{j-1}, Q_{x_{j+1}}^{j+1}, \dots, Q_{x_k}^k \ M[x_1, \dots, x_{j-1}, x_j/f_{x_j}(x_1, \dots, x_{j-1}), x_{j+1}, \dots, x_k]$$

a fórmula resultante do primeiro passo da Skolemização.

Suponha que F é inconsistente. Se F_{Sk_1} é consistente então existe uma interpretação \mathcal{I} modelo de F_{Sk_1} . A interpretação \mathcal{I} designa para o símbolo f_{x_j} uma função $f^{\mathcal{I}}$ $(j-1)$ -ária tal que para toda $(j-1)$ -tupla no domínio D de \mathcal{I} , d_1, \dots, d_{j-1} existe um elemento em D , que é $f^{\mathcal{I}}(d_1, \dots, d_{j-1})$, tal que

$$Q_{x_{j+1}}^{j+1}, \dots, Q_{x_k}^k \ M[x_1/d_1, \dots, x_{j-1}/d_{j-1}, x_j/f^{\mathcal{I}}(d_1, \dots, d_{j-1}), x_{j+1}, \dots, x_k]$$

é verdadeira em \mathcal{I} . Consequentemente F_{fnp} é verdadeira em \mathcal{I} o que contradiz a suposição inicial.

Suponha agora que F_{Sk_1} é inconsistente. Se F é consistente, então existe um modelo \mathcal{I} de F_{fnp} . A interpretação \mathcal{I} pode-se estender de maneira tal que inclui uma função f que aplica cada $(j-1)$ -tupla d_1, \dots, d_{j-1} do domínio D da interpretação \mathcal{I} a um elemento d de D tal que a fórmula

$$Q_{x_{j+1}}^{j+1}, \dots, Q_{x_k}^k \ M[x_1/d_1, \dots, x_{j-1}/d_{j-1}, x_j/d, x_{j+1}, \dots, x_k]$$

é verdadeira em \mathcal{I} ou identicamente a fórmula

$$Q_{x_{j+1}}^{j+1}, \dots, Q_{x_k}^k \ M[x_1/d_1, \dots, x_{j-1}/d_{j-1}, x_j/f(d_1, \dots, d_{j-1}), x_{j+1}, \dots, x_k]$$

é verdadeira em \mathcal{I}' . Ao definir f desta maneira temos que a fórmula F_{Sk_1} é verdadeira em \mathcal{I}' o que contradiz a suposição inicial. \square

Observação: Não se pode falar de “equivalência” entre a lógica de primeira-ordem e a clausal; o que se é certo é que o processo de Skolemização (das formas normais conjuntivas) de uma sentença da lógica de primeira-ordem que seja inconsistente gerará uma fórmula clausal que deve ser também inconsistente (e vice versa). Assim, se o problema é testar a inconsistência de sentenças da lógica de primeira-ordem, basta tratar o problema na lógica clausal. \bullet

Exemplo 1.4.4 Para demonstrar que uma sentença F não é necessariamente equivalente a sua Skolemização F_{Sk} considere a fórmula $\exists_x p(x)$ e sua Skolemização $p(a)$. Seja \mathcal{I} a interpretação sobre o domínio $D = \{0, 1\}$ que designa para a o elemento 0, para $p(0)$ *false* e para $p(1)$ *true*. Claramente a fórmula é verdadeira em \mathcal{I} , mas sua Skolemização não. \diamond

1.5 Resolução no cálculo proposicional

Neste ponto da discussão é possível ilustrar o uso do princípio de resolução com exemplos do cálculo ou lógica proposicional; i.e., teorias axiomatizadas com a linguagem da lógica de primeira ordem usando unicamente símbolos de predicado zero-ários.

Exemplo 1.5.1 Sejam p, q, r símbolos proposicionais. Deseja-se provar que $p \wedge q \wedge r$ é consequência lógica de $S = \{p \wedge (\neg r \vee p), r, \neg r \vee \neg p \vee q\}$. Basta então demonstrar a inconsistência do conjunto $\{\neg(p \wedge q \wedge r)\} \cup \{p \wedge (\neg r \vee p), r, \neg r \vee \neg p \vee q\}$

Observe que como r é verdadeiro e igualmente p o é, já que $p \wedge (\neg r \vee p)$ o é, então q é também verdadeiro, já que $\neg r \vee \neg p \vee q \equiv q \leftarrow r, p$ o é. Consequentemente, $\neg(p \wedge q \wedge r)$ é falsa em qualquer modelo de S . \diamond

A resolução no cálculo proposicional aplica-se para cláusulas gerais; i.e., disjunções de símbolos proposicionais e/ou suas negações. A operacionalização do princípio de resolução no cálculo proposicional basea-se na aplicação de duas regras de inferência denominadas **regra de corte** e **regra de simplificação** sobre cláusulas (gerais).

Definição 1.5.2 (Regra de corte) *Sejam $p \vee l_1 \vee \dots \vee l_n$ e $\neg p \vee l'_1 \vee \dots \vee l'_m$ duas cláusulas proposicionais. Então a cláusula $l_1 \vee \dots \vee l_n \vee l'_1 \vee \dots \vee l'_m$ é inferida das duas anteriores pela **regra de corte**.*

Definição 1.5.3 (Regra de simplificação) *Seja $p \vee p \vee l_1 \vee \dots \vee l_n$ uma cláusula proposicional. Então a cláusula $p \vee l_1 \dots l_n$ é inferida da anterior pela **regra de simplificação**. Igualmente $\neg p \vee l_1 \dots l_n$ infere-se de $\neg p \vee \neg p \vee l_1 \vee \dots \vee l_n$ pela regra de simplificação.*

A notação usual para representar as regras de inferência de corte e simplificação é a seguinte:

$$\frac{D \vee p, C \vee \neg p}{D \vee C} \quad \text{onde } D \text{ e } C \text{ são cláusulas gerais} \quad \text{Regra de corte}$$

$$\frac{D \vee p \vee p}{D \vee p} \quad \text{onde } D \text{ é cláusula geral} \quad \text{Regra de simplificação}$$

$$\frac{D \vee \neg p \vee \neg p}{D \vee \neg p} \quad \text{onde } D \text{ é cláusula geral} \quad \text{Regra de simplificação}$$

Uma prova refutacional pelo método de resolução no cálculo proposicional consiste, então, de uma sequência de aplicações das anteriores regras de inferência até atingir a cláusula vazia.

Observe que a regra de corte “ $D \vee C$ se infere de $D \vee p$ e $C \vee \neg p$ ” pode-se assimilar equivalentemente da seguinte maneira: “ $\leftarrow \neg D, \neg C$ infere-se aplicando a regra $p \leftarrow \neg D$ ao objetivo $\leftarrow p, \neg C$ ”.

Exemplo 1.5.4 (Continuação do exemplo 1.5.1) Observe primeiro que o conjunto $\{p \wedge (\neg r \vee p), r, \neg r \vee \neg p \vee q\}$ de fórmulas proposicionais se pode transformar no seguinte conjunto equivalente de cláusulas de programa $P = \{p, p \leftarrow r, r, q \leftarrow r, p\}$. Para refutar com este programa o objetivo $\{\neg(p \wedge q \wedge r) \equiv \leftarrow p, q, r\}$, aplicam-se as regras de corte e simplificação para o conjunto equivalente de cláusulas gerais obtido construindo as correspondentes formas normais conjuntivas das cláusulas:

$$\{\neg p \vee \neg q \vee \neg r, p, \neg r \vee p, r, \neg r \vee \neg p \vee q\}$$

Assim, aplicando a regra de corte, de $\neg p \vee \neg q \vee \neg r$ e p obtém-se $\neg q \vee \neg r$;

aplicando a regra de corte, de $\neg q \vee \neg r$ e $\neg r \vee \neg p \vee q$ obtém-se $\neg r \vee \neg r \vee \neg p$;

aplicando a regra de simplificação, obtém-se $\neg r \vee \neg p$;

aplicando a regra de corte, do anterior e r obtém-se $\neg p$;

aplicando a regra de corte, de $\neg p$ e p obtém-se finalmente a cláusula vazia, \square , completando a prova.

Observe que existem outras possíveis provas mais simples. \diamond

Observação: As provas pelo método de resolução podem ser ilustradas como árvores (invertidas), onde cada vértice corresponde a uma cláusula, as folhas estão marcadas com cláusulas do conjunto original e os antecessores de uma cláusula correspondem às premissas utilizadas na sua inferência seja por regra de corte (dois antecessores) ou por simplificação (um antecessor). Uma representação de uma refutação tem raiz correspondente à cláusula vazia. \bullet

Exemplo 1.5.5 (Continuação do exemplo 1.5.4) A prova pode ser apresentada na estrutura de árvore de inferências como na figura 1.1.

Para outras possíveis provas a estrutura da árvore de inferência mudará. \diamond

O método de resolução no cálculo proposicional é *correto* no sentido de que qualquer cláusula C que se deduz aplicando regras de corte e simplificação a partir de um conjunto S de cláusulas proposicionais é uma consequência lógica de S .

Correção e completeude, termos previamente utilizados sem explicação, são noções básicas da lógica formal. Dada uma teoria, possivelmente apresentada com axiomas lógicos, diz-se que um método de dedução é *correto*, quando a aplicação do dito método permite demonstrar unicamente consequências lógicas da teoria. Diz-se que um método dedutivo é *completo*, quando, além de ser

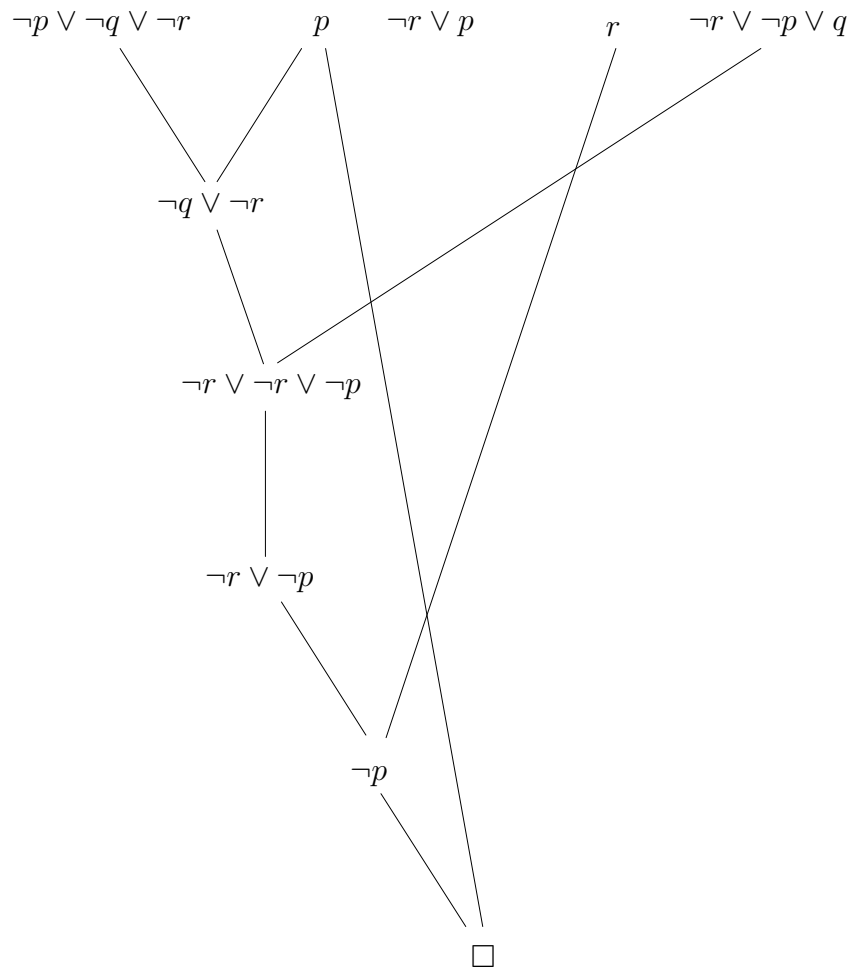


Figura 1.1: Árvore de uma refutação por resolução

correto, todas as consequências lógicas da teoria podem ser demonstradas com a sua aplicação. Os seguintes lemas expressam a correção e uma versão restrita da completude do método de resolução no cálculo proposicional.

Lema 1.5.6 (Correção da resolução proposicional) *O método de resolução no cálculo proposicional é correto.*

Demonstração. Basta provar que cada uma das regras de inferência é *correta*. Obviamente, qualquer modelo de $p \vee p \vee D$ é modelo de $p \vee D$ e igualmente para $\neg p \vee \neg p \vee D$ e $\neg p \vee D$. Por outra parte qualquer modelo de $D \vee p$ e $C \vee \neg p$ é modelo de $D \vee C$, já que não é possível que tanto p como $\neg p$ sejam ambos verdadeiros no modelo. \square

O outro sentido do lema, a *completude* da resolução proposicional, não é verdadeiro quando a resolução é utilizada diretamente. Veja exercício 1.11.

Lema 1.5.7 (Completeness refutacional da resolução proposicional) *O método de resolução no cálculo proposicional é refutacionalmente completo; i.e., um conjunto de cláusulas proposicionais S é instatisfável sse a cláusula vazia pode ser deduzida de S pela aplicação das regras de corte e simplificação.*

Demonstração. (Idéia) Pela correção temos que se \square se deduz de S então S é inconsistente. No outro sentido demonstre primeiro que se S é inconsistente e o conjunto de símbolos proposicionais que ocorrem nas cláusulas de S é $\{p_1, \dots, p_n\}$ então é possível obter de S por resolução um conjunto de cláusulas S' tal que unicamente os símbolos proposicionais $\{p_1, \dots, p_{n-1}\}$ ocorrem em S' . A demonstração segue então por indução no número de símbolos proposicionais que ocorrem em S . Os detalhes são deixados como exercício 1.12. \square

Observe que qualquer fórmula válida F do cálculo proposicional pode ser demonstrada indiretamente utilizando resolução: basta computar a forma normal conjuntiva de $\neg F$, que é um conjunto de cláusulas inconsistente. Assim, utilizando o lema de completeness refutacional este conjunto de cláusulas deverá ter uma dedução da cláusula vazia via resolução.

Exercícios do Capítulo 1

Exercício 1.1 Considere a interpretação \mathcal{I} com domínio \mathbb{N} e as seguintes designações:

- para o símbolo de função unária s corresponde a função sucessor ($x \mapsto x + 1$);
- para o símbolo de constante a corresponde 0;
- para o símbolo de constante b corresponde 1;
- para o símbolo de predicado binário p corresponde a relação $\{(x, y) \mid x > y\}$;
- para o símbolo de predicado unário q corresponde a relação $\{x \mid x > 0\}$;
- para o símbolo de predicado binário r corresponde a relação $\{(x, y) \mid x \text{ divide } y\}$.

q Determine o *valor de verdade* das seguintes sentenças:

- a. $\forall x \exists y p(x, y)$,
- b. $\exists x \forall y p(x, y)$,
- c. $p(s(a), b)$,
- d. $\forall x (q(x) \rightarrow p(x, a))$,
- e. $\forall x p(s(x), x)$,
- f. $\forall x \forall y (r(x, y) \rightarrow \neg p(x, y))$,
- g. $\forall x (\exists y p(x, y) \vee r(s(b), s(x)) \rightarrow q(x))$.

Exercício 1.2 Detemine se as seguintes fórmulas são ou não *válidas*:

- a. $\forall x \exists y p(x, y) \rightarrow \exists y \forall x p(x, y)$,
- b. $\exists y \forall x p(x, y) \rightarrow \forall x \exists y p(x, y)$.

Exercício 1.3 Considere a fórmula:

$$(\forall x p(x, x) \wedge \forall x \forall y \forall z ((p(x, y) \wedge p(y, z)) \rightarrow p(x, z)) \wedge \forall x \forall y (p(x, y) \vee p(y, x))) \rightarrow \exists y \forall x p(y, x)$$

- Demonstre que cada interpretação com um domínio finito é um *modelo* da fórmula.

- Encontre uma interpretação que não seja modelo da fórmula.

Exercício 1.4 Seja F uma sentença. Demonstre que existe uma fórmula G que é uma conjunção de cláusulas que satisfaz o seguinte: G é *inconsistente* sse F é *inconsistente*.

Exercício 1.5 Usando a lógica de primeira ordem simbolizar as seguintes expressões:

- João vota unicamente em políticos honestos.
- Alguns políticos não votam neles mesmos.
- Um político não é necessariamente honesto porque todos votam nele.
- O barbeiro barbeia a todos aqueles que não se barbeiam.
- João pode enganar a alguma gente todos os dias e pode enganar toda a gente alguns dias, mas não pode enganar a todos todos os dias. (Use um símbolo de predicado 3-ário).

Exercício 1.6 Suponha $p(x, y)$ é interpretado como “ x é pai de y ” e $m(x, y)$ como “ x é mãe de y ”. c e d são constantes interpretadas como João e Pedro, respectivamente. Identifique as relações familiares expressadas nas seguintes fórmulas:

- $\exists_x \exists_y (m(x, c) \wedge m(x, d) \wedge p(y, c) \wedge p(y, d))$,
- $\exists_x \exists_y \exists_z (p(x, y) \wedge p(x, z) \wedge m(y, c) \wedge p(z, d))$,
- $\exists_x (p(c, x) \wedge m(x, d))$,
- $\exists_x (p(c, x) \wedge p(x, d))$,
- $p(c, d) \wedge \forall_x (p(c, x) \rightarrow x = d)$.

São *válidas* as seguintes fórmulas:

- $\forall_x \exists_y (m(x, y) \wedge \forall_z (m(z, x) \rightarrow z = y))$,
- $\neg \exists_x p(x, x)$.

Lembre-se que a noção de validade independe de uma interpretação específica.

Exercício 1.7 É possível obter sentenças verdadeiras em $\langle \mathbb{Q}, < \rangle$ e falsas em $\langle \mathbb{R}, < \rangle$ ou *vice versa*? $\langle \mathbb{Q}, < \rangle$ e $\langle \mathbb{R}, < \rangle$ denotam respectivamente os racionais e reais com a relação menor.

Exercício 1.8 Escreva uma sentença que não tenha *modelos finitos* (i.e., interpretações com domínio finito).

Exercício 1.9 Escreva uma sentença diferente da apresentada no exercício 1.3 que seja igualmente verdadeira em qualquer modelo finito, mas falsa em alguma interpretação infinita.

Exercício 1.10 Converta as seguintes fórmulas à *forma normal prenexa conjuntiva*:

a. $\forall_x p(x) \leftrightarrow \exists_x q(x)$,

b. $\forall_x r(x, y) \vee \forall_x q(x, x)$,

c. $\neg \exists_x (\forall_y r(x, y) \rightarrow \neg \exists_y \forall_w s(y, w, x))$,

d. $\exists_y \forall_x r(y, x) \rightarrow \forall_w \forall_y r(w, y)$,

e. $\forall_x (2 < x \wedge \forall_y \forall_z (y \cdot z = x \rightarrow (y = x \vee y = 1))) \rightarrow \exists_y ((y + y) + 1 = x)$.

“todo primo maior que 2 é ímpar”.

Exercício 1.11 Demonstre a *incompletude* do método de resolução no cálculo proposicional. Basta encontrar duas cláusulas C, D tais que $C \rightarrow D$ seja válida, mas seja impossível deduzir diretamente D de C pelas regras de corte e simplificação. Observe que indiretamente é possível sempre demonstrar qualquer fórmula proposicional válida.

Exercício 1.12 Complete a prova da completude da resolução no cálculo proposicional, lema 1.5.7.

Exercício 1.13 Demonstre usando o método de resolução no cálculo proposicional que as seguintes fórmulas são consequências lógicas dos conjuntos de fórmulas correspondentes

a. $p \wedge (q \rightarrow r), \{(p \rightarrow q) \rightarrow r, r \rightarrow p\}$;

b. $\neg r \rightarrow s, \{p \rightarrow (q \rightarrow r), (q \wedge p) \vee s\}$;

c. $\neg q \rightarrow t, \{p \rightarrow (\neg q \wedge r), q \vee s, (p \rightarrow u) \rightarrow \neg s, r \rightarrow (p \wedge t)\}$.